Testing en Java

El testing en Java es una parte esencial del desarrollo de software y sirve para garantizar que el código que escribimos funcione correctamente, sea confiable y cumpla con ciertos requisitos.

La forma de corroborar esto es a través de la realización de pruebas y para ello existen diferentes tipos de pruebas en Java.

Pruebas Unitarias: Son un tipo de pruebas que se enfocan en verificar el funcionamiento correcto de componentes individuales del código (como métodos y clases) de manera aislada. El propósito de las pruebas unitarias es asegurarse de que cada unidad funcione según lo esperado y produzca resultados correctos, independientemente de cómo se integren con otros componentes. Para realizar estas pruebas se usan frameworks como JUnit.

Pruebas de Integración: Son un tipo de pruebas que se centran en verificar la interacción y la colaboración correcta entre diferentes componentes de un sistema o aplicación. En contraste con las *pruebas unitarias*, que se centran en probar unidades individuales de código de manera aislada, las pruebas de integración evalúan cómo los módulos o componentes interactúan cuando se combinan. Las pruebas de integración son esenciales para garantizar que las diversas partes de una aplicación trabajen juntas de manera coherente y cumplan con los requisitos y expectativas del sistema en su conjunto.

Pruebas de Regresión: Son un tipo de pruebas que se realizan para asegurarse de que las modificaciones recientes en el código no hayan afectado negativamente las funcionalidades existentes del software. Es decir, las pruebas de regresión buscan identificar cualquier regresión o retroceso en la calidad del software debido a cambios nuevos o actualizaciones.

Cuando se introducen nuevas características, se corrigen errores o se realizan cambios en el código existente, existe la posibilidad de que estos cambios puedan afectar áreas no relacionadas del software y provocar fallos en funcionalidades previamente probadas y funcionales. Las pruebas de regresión se llevan a cabo para mitigar este riesgo y garantizar que las funcionalidades existentes continúen funcionando como se esperaba. Las pruebas de regresión son fundamentales para mantener la integridad y la calidad del software a medida que evoluciona con el tiempo. Al automatizar las pruebas de regresión, se puede reducir el esfuerzo manual requerido y asegurarse de que las funcionalidades existentes se mantengan sin problemas en cada iteración del desarrollo.

Pruebas de Aceptación: Son un tipo de pruebas que se realizan para validar que el software cumple con los requisitos y expectativas del usuario y del negocio. Estas pruebas se centran en verificar que el software funcione de acuerdo con los criterios de

aceptación definidos previamente y que se ajuste a los casos de uso y las necesidades de los usuarios finales.

Las pruebas de aceptación son esenciales para garantizar que el *software entregado* sea útil, confiable y satisfaga las necesidades reales de los usuarios y las partes interesadas. A menudo, estas pruebas se realizan al final del ciclo de desarrollo, después de que las pruebas unitarias y las pruebas de integración hayan sido completadas. Pueden ser realizadas manualmente o de manera automatizada, dependiendo del contexto y la naturaleza de las pruebas. En otras palabras, estas pruebas ayudan a validar que el software funcione según lo previsto y es un paso crítico antes de su lanzamiento.

Frameworks de pruebas

Los frameworks de pruebas en Java son herramientas esenciales para la práctica de pruebas efectivas en el desarrollo de software. Un framework de pruebas es una infraestructura y un conjunto de herramientas diseñadas para facilitar la creación, organización, ejecución y gestión de pruebas de software en proyectos de Java. Los frameworks de pruebas proporcionan un marco de trabajo estructurado y coherente que posee un conjunto de funcionalidades las cuales permiten a los desarrolladores escribir y ejecutar pruebas de manera efectiva, lo que ayuda a mantener la calidad y la confiabilidad del software a lo largo del tiempo y facilita la identificación temprana de errores y la validación del comportamiento del software en diferentes situaciones.

Las herramientas más utilizadas para hacer testing en Java son:

JUnit: Es un framework de pruebas unitarias para Java, y su principal propósito es proporcionar una estructura y un conjunto de herramientas que permitan a los desarrolladores escribir, organizar y ejecutar pruebas unitarias de manera efectiva.

Las principales características y herramientas de *JUnit* son:

- **Anotaciones en JUnit:**
  JUnit utiliza anotaciones para identificar métodos de prueba y configurar el comportamiento de las pruebas. Algunas anotaciones importantes son:
  - ✓ **@Test:** Marca un método como un método de prueba.
  - ✓ **@Before:** Indica un método que se ejecutará antes de cada método de prueba.
  - ✓ **@After:** Indica un método que se ejecutará después de cada método de prueba.
  - ✓ **@BeforeClass:** Indica un método que se ejecutará una vez antes de todas las pruebas en la clase.
  - ✓ **@AfterClass:** Indica un método que se ejecutará una vez después de todas las pruebas en la clase.
- **Asserts en JUnit:**
  JUnit proporciona una variedad de *métodos assert* que permiten verificar las condiciones esperadas en las pruebas. Algunos ejemplos son:

- **assertEquals:** Verifica que los valores esperados y actuales sean iguales.
- **assertTrue:** Verifica que la condición proporcionada sea verdadera.
- **assertFalse:** Verifica que la condición proporcionada sea falsa.
- **assertNull:** Verifica que el objeto proporcionado sea nulo.
- **assertNotNull:** Verifica que el objeto proporcionado no sea nulo.

Integración con Herramientas de Construcción y Entornos de Desarrollo:

JUnit se integra bien con herramientas de construcción como Maven y Gradle, lo que facilita la ejecución de pruebas como parte del proceso de compilación. Además, muchas IDEs como Eclipse e IntelliJ IDEA ofrecen soporte integrado para la creación y ejecución de pruebas JUnit.

Selenium: Es una herramienta esencial para la automatización de pruebas funcionales en aplicaciones web. Ayuda a mejorar la eficiencia y la cobertura de pruebas, y permite a los equipos de desarrollo mantener la calidad y la confiabilidad de las aplicaciones web a medida que evolucionan con el tiempo. Selenium es especialmente útil para probar aplicaciones web en diferentes navegadores y sistemas operativos.

Las funciones y beneficios de Selenium son:

- Permite automatizar interacciones en un navegador web, como hacer clic en botones, llenar formularios, navegar por páginas y verificar resultados.
- Es compatible con varios navegadores populares, incluyendo Chrome, Firefox, Edge, Safari y más. Esto permite probar la funcionalidad en diferentes navegadores.
- Las pruebas automatizadas con Selenium ofrecen una reproducción consistente de interacciones, lo que minimiza los errores humanos y garantiza resultados confiables.
- Permite verificar que una aplicación web funcione correctamente en múltiples navegadores y versiones.
- Puede ser utilizado para probar aplicaciones web en diferentes sistemas operativos, como Windows, macOS y Linux.
- Es ideal para realizar pruebas de regresión, asegurando que las nuevas modificaciones en el código no afecten el funcionamiento de la aplicación web existente.

Cypress: es una herramienta de automatización de pruebas diseñada específicamente para realizar pruebas en aplicaciones web modernas. A diferencia de Selenium u otras herramientas similares, Cypress ofrece una experiencia de automatización más rápida y eficiente al proporcionar un conjunto integrado de características que facilitan la escritura y ejecución de pruebas funcionales y de integración en aplicaciones web.

Las funciones y beneficios de Cypress son:

- Permite ver en tiempo real cómo las acciones automatizadas interactúan con la aplicación a medida que se ejecutan las pruebas, lo que facilita la depuración y la comprensión del comportamiento de la aplicación.
- Se ejecuta en el mismo contexto que la aplicación web, lo que significa que puede interactuar directamente con el DOM y las APIs de la aplicación, lo que simplifica la escritura de pruebas.
- Proporciona una ventana de visualización que muestra el estado en tiempo real de la aplicación durante la ejecución de las pruebas, lo que facilita la identificación de problemas.
- Utiliza una sintaxis declarativa para escribir pruebas, lo que hace que el código sea más legible y comprensible.
- Es adecuado tanto para pruebas simples de un solo componente como para pruebas más complejas de flujos de trabajo y escenarios de usuario.
- Se integra bien con herramientas de desarrollo, como Chrome DevTools, lo que permite una depuración más efectiva.
- Puede utilizarse tanto para pruebas de integración que involucran múltiples componentes como para pruebas de unidad a nivel de componente.
- Puede integrarse con herramientas de integración continua y entrega continua (CI/CD) para ejecutar pruebas automatizadas en cada confirmación de código.

Cucumber: Es un framework de pruebas de comportamiento o BDD (Behavior-Driven Development) diseñado para mejorar la colaboración entre desarrolladores, testers y partes interesadas no técnicas en el proceso de desarrollo de software. A diferencia de los frameworks de pruebas unitarias tradicionales, Cucumber se centra en la descripción de comportamientos del software en un lenguaje natural y permite la automatización de estas especificaciones en forma de pruebas automatizadas.

Cucumber utiliza el *«lenguaje Gherkin»* para describir el comportamiento del software en un formato legible por humanos. Gherkin utiliza palabras clave como *"Given" (Dado)*, *"When" (Cuando)* y *"Then" (Entonces)* para definir los pasos de las pruebas.

Sonar: La *plataforma SonarQube* (comúnmente referida como "Sonar") es una herramienta de análisis de código estático y gestión de calidad del software. Su objetivo principal es evaluar la calidad del código fuente de un proyecto de software y proporcionar retroalimentación sobre áreas que requieren mejoras. Sonar ayuda a los equipos de desarrollo a mantener la integridad, la confiabilidad y la eficiencia del código a lo largo del ciclo de vida del software.

Las principales funciones y beneficios de Sonar son:

- Realiza un análisis en profundidad del código fuente para identificar problemas de calidad, como violaciones de estándares de codificación, vulnerabilidades, duplicaciones y complejidad excesiva.
- Proporciona métricas y puntuaciones de calidad del código, como la cobertura de pruebas, la deuda técnica y la complejidad del código, lo que ayuda a evaluar la salud del proyecto.
- Puede identificar vulnerabilidades de seguridad y errores de programación en el código, ayudando a prevenir problemas de seguridad y fallas en la aplicación.

Mockito: Es un framework de Java que se utiliza para crear objetos simulados o *"mocks"* en pruebas unitarias. Su principal objetivo es facilitar el aislamiento de unidades de código (como clases y métodos) durante las pruebas, permitiendo que estas unidades se prueben de manera independiente de sus dependencias externas. Mockito es ampliamente utilizado en el desarrollo de pruebas unitarias para simular el comportamiento de componentes externos y controlar las interacciones entre objetos.

PiTest: Ayuda a identificar áreas de código que no están siendo cubiertas adecuadamente por las pruebas a partir de *mutaciones en el código*. PiTest es una herramienta poderosa para evaluar la calidad y la cobertura de las pruebas unitarias en Java mediante la técnica de mutación. Ayuda a identificar áreas de mejora en el conjunto de pruebas y a fortalecer la confiabilidad y la efectividad del software.