**1. Diferencias entre pruebas unitarias, de integración y de extremo a extremo (E2E):**

* **Pruebas Unitarias:** Evalúan partes específicas de una aplicación, generalmente funciones o métodos individuales, en aislamiento.

**Ejemplo:** Probar una función que calcula impuestos.

**Objetivo:** Garantizar que cada componente funciona según lo esperado de forma independiente.

* **Pruebas de Integración:** Verifican cómo interactúan los diferentes módulos o componentes de la aplicación.

**Ejemplo:** Comprobar que un módulo de autenticación puede comunicarse con una base de datos.

**Objetivo:** Identificar problemas en la interacción entre componentes.

**Pruebas de Extremo a Extremo (E2E):** Simulan el flujo completo de la aplicación desde la perspectiva del usuario final.

**Ejemplo**: Probar un proceso de compra en un sitio web desde agregar al carrito hasta el pago.

**Objetivo**: Validar que todo el sistema funciona de manera integrada.

**2. ¿Qué es un framework de automatización de pruebas? Ejemplos utilizados:**

Un framework de automatización de pruebas es un conjunto de herramientas, librerías y directrices que facilitan la creación, ejecución y mantenimiento de pruebas automatizadas. Proporciona estructura y consistencia a los scripts de prueba.

**Ejemplos utilizados:**

**Selenium:** Para pruebas web.

**Serenity BDD:** Gestión de pruebas con reportes detallados.

**Cucumber:** Automatización basada en lenguaje Gherkin.

**Karate:** Automatización de pruebas de APIs.

**3. Concepto de "Pirámide de Pruebas" y su importancia:**

La Pirámide de Pruebas es un modelo que sugiere la distribución ideal de pruebas en una aplicación:

* **Base:** Pruebas unitarias (mayor número, bajo costo y rápidas).
* **Medio:** Pruebas de integración (menos que unitarias, más que E2E).
* **Cima:** Pruebas E2E (pocas, lentas y costosas).

**Importancia:**

* Reduce costos y tiempos de prueba.
* Proporciona una cobertura adecuada en diferentes niveles.
* Ayuda a detectar errores tempranamente.

**4. Sincronización en pruebas automatizadas y su importancia:**

La sincronización asegura que las pruebas automatizadas esperen eventos específicos antes de interactuar con los elementos del sistema, como la carga de una página o la aparición de un elemento.

**Importancia:**

* Evita fallos en las pruebas por tiempos incorrectos.
* Mejora la estabilidad y confiabilidad de las pruebas.
* **Ejemplo:** Usar comandos como wait o condiciones explícitas en Selenium para esperar a que un botón sea clickeable.

**5. Integración de pruebas automatizadas en un proceso de CI:**

* Configurar un servidor CI (como Jenkins o GitHub Actions).
* Crear scripts que ejecuten las pruebas automáticamente tras cada commit.
* Integrar las pruebas en el pipeline de CI con herramientas como Maven o Gradle.
* Generar reportes de resultados para retroalimentación inmediata.
* Detecta errores rápidamente, asegurando calidad continua.

**6. Diferencias entre pruebas de carga y estrés:**

**Pruebas de Carga:**

* Evalúan el desempeño del sistema bajo condiciones normales y máximas de uso esperado.
* **Ejemplo:** Simular 1000 usuarios accediendo simultáneamente a un sitio web.
* **Objetivo:** Determinar el rendimiento y capacidad del sistema.

**Pruebas de Estrés:**

Empujan al sistema más allá de sus límites normales.

**Ejemplo:** Simular 5000 usuarios en un servidor diseñado para 1000.

**Objetivo:** Identificar puntos de falla y cómo el sistema se recupera.

**7. Gestión de pruebas automatizadas para aplicaciones con componentes dinámicos:**

* Utilizar localizadores robustos (como XPath dinámico o identificadores únicos).
* Implementar estrategias de sincronización (esperas explícitas o condiciones de aparición).
* Usar técnicas de reestructuración del código para manejar elementos dinámicos (como Page Object Model).
* Incorporar validaciones condicionales para manejar cambios en tiempo de ejecución.

**8. Importancia de gestionar dependencias en automatización de pruebas y cómo hacerlo:**

**Importancia:**

* Facilita el uso de bibliotecas y herramientas necesarias para las pruebas.
* Evita conflictos de versiones.
* Simplifica el mantenimiento del proyecto.

**Cómo hacerlo:**

**Maven:** Usar el archivo pom.xml para definir dependencias con versiones específicas.

**Gradle:** Configurar dependencias en build.gradle usando sintaxis como implementation o testImplementation.