Icono *Training Consulting*

Formación

SOAP UI para verificar Web Services

2025

Formador

Ana Isabel Vegas

INGENIERA INFORMÁTICA con Master Máster Universitario en Gestión y Análisis de Grandes Volúmenes de Datos: Big Data, tiene la certificación PCEP en Lenguaje de Programación Python y la certificación JSE en Javascript. Además de las certificaciones SCJP Sun Certified Programmer for the Java 2 Platform Standard Edition, SCWD Sun Certified Web Component Developer for J2EE 5, SCBCD Sun Certified Business Component Developer for J2EE 5, SCEA Sun Certified Enterprise Architect for J2EE 5.

Desarrolladora de Aplicaciones FULLSTACK, se dedica desde hace + de 20 años a la CONSULTORÍA y FORMACIÓN en tecnologías del área de DESARROLLO y PROGRAMACIÓN.

training@iconotc.com

SOAP UI para verificar Web Services

❑ **Duración:** 18 horas

❑ **Modalidad:** On-line

❑ **Fechas/Horario:**

- Días 2, 3 y 4 de Abril 2025
- Horario de 9:00 a 15:00 hs

❑ **Contenidos:**

- Realización de pruebas web incluyendo métodos de captura y repetición de pruebas.
- Monitorización y control de acciones web.
- Grabación de scripts.
- Grabación de estadísticas mientras se ejecuta una prueba.
- Configuración: Project, TestSuite,TestCase, Steps.
- Testing manual con SOAP: Assertions , Propiedades y Property Transfer. Operaciones con request y response.
- Testing manual con REST.
- Pruebas síncronas y asíncronas.
- Automatización de pruebas (ant, maven, jenkins).
- Servicios Mock; con SOAP Y REST.
- Groovy básico.
- Informes con SoapUI PRO.
- Pruebas de rendimiento: Creación y análisis de resultados.
- Security Test: Creación y análisis de resultados.

Introducción a SOAP UI

Tema 0

SoapUI

- SoapUI es una herramienta líder de código abierto para realizar pruebas de API, especialmente diseñada para servicios web SOAP y REST.
- Desarrollada en Java, SoapUI ofrece una interfaz de usuario intuitiva que permite a usuarios técnicos y no técnicos realizar diversas pruebas de forma eficiente
- Descarga del sitio web: https://www.soapui.org/downloads/soapui/

ReadyAPI

Get the most advanced functional testing tool for REST,

SOAP and GraphQL APIs.

Download ReadyAPI

Características principales de SoapUI

- **Pruebas funcionales**: Permite escribir y ejecutar pruebas automatizadas para verificar la funcionalidad de las API.
- **Pruebas de carga**: Facilita la evaluación del rendimiento de las API bajo diferentes condiciones de carga.
- **Pruebas de seguridad**: Ayuda a identificar vulnerabilidades en las API.
- **Soporte multiplataforma**: Compatible con varios protocolos como SOAP, REST, HTTP, JMS, JDBC y más.
- **Automatización**: Permite la creación de scripts de prueba y su integración en procesos de integración continua

Ventajas de SoapUI

- **Interfaz intuitiva**: Fácil de usar para principiantes y expertos.
- **Versatilidad**: Soporta múltiples protocolos y tipos de pruebas.
- **Código abierto**: Disponible gratuitamente con una gran comunidad de soporte.
- **Eficiencia**: Permite ejecutar pruebas completas en poco tiempo.
- **Integración**: Compatible con herramientas de terceros como Git y Jenkins

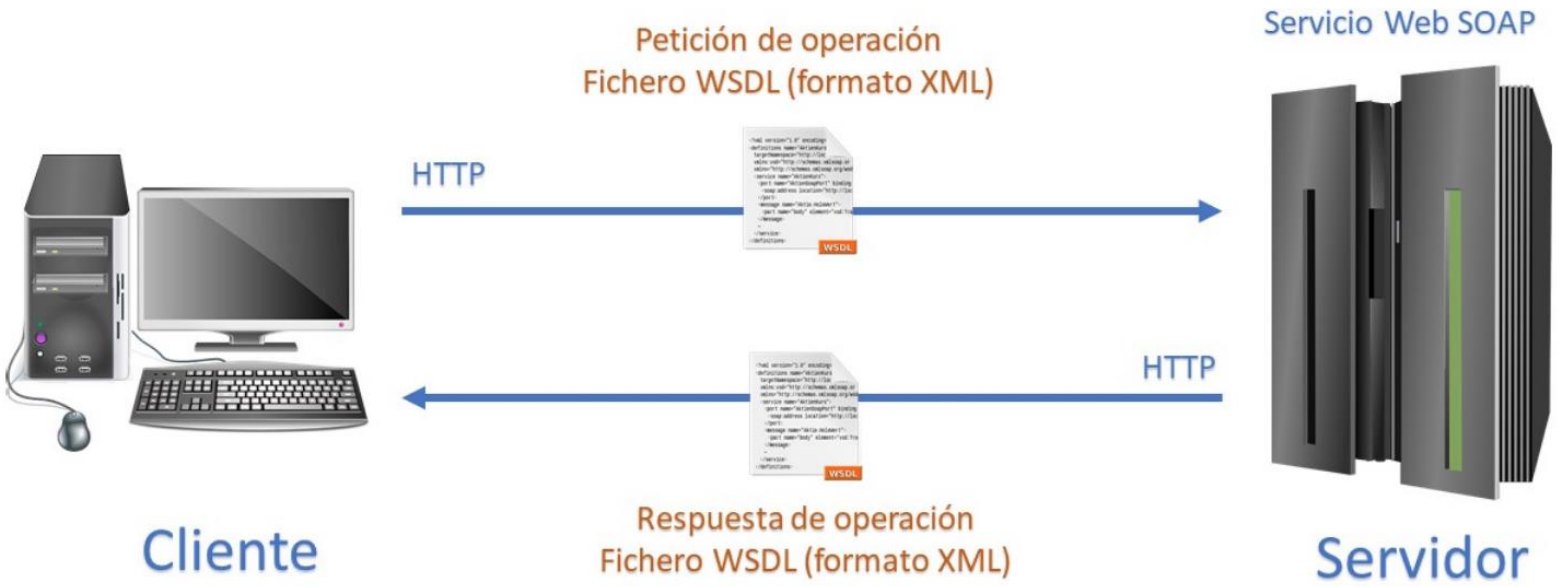
Servicios SOAP

- Un servicio **SOAP (Simple Object Access Protocol)** tiene las siguientes características principales:
- SOAP **utiliza exclusivamente XML** para estructurar los mensajes de solicitud y respuesta, lo que garantiza la interoperabilidad entre diferentes sistemas y plataformas.
- Un mensaje SOAP tiene una estructura bien definida:
  - **Envelope** (Sobre): Es el elemento raíz que identifica al mensaje como un mensaje SOAP.
  - **Header** (Cabecera): Es opcional y se utiliza para incluir información adicional, como metadatos o configuraciones específicas del mensaje.
  - **Body** (Cuerpo): Es obligatorio y contiene los datos principales de la solicitud o respuesta.
  - **Fault** (Error): Es un subelemento del cuerpo utilizado para notificar errores durante el procesamiento.
- SOAP puede funcionar sobre **diferentes protocolos de transporte**, como HTTP, SMTP, TCP o JMS. Sin embargo, HTTP es el más común debido a su compatibilidad con la infraestructura web existente.
- Los servicios SOAP suelen estar acompañados por un archivo **WSDL** que describe las operaciones disponibles, los parámetros requeridos y las respuestas esperadas.

Servicios SOAP

- Permite agregar **funcionalidades adicionales mediante extensiones**, como WS-Security para seguridad avanzada o WS-ReliableMessaging para garantizar la entrega confiable de mensajes.
- **Es independiente del lenguaje de programación y la plataforma**, lo que facilita la comunicación entre sistemas heterogéneos.
- SOAP soporta estándares avanzados como **WS-Security,** que proporcionan autenticación, integridad y confidencialidad en los mensajes. Esto lo hace ideal para aplicaciones empresariales críticas, como servicios financieros o de salud.
- Soporte para **RPC y Estilo Documental**. SOAP admite dos estilos principales:
  - Llamada a Procedimiento Remoto (RPC): Para invocar métodos en el servidor.
  - Estilo Documental: Más flexible y orientado a mensajes, útil para transferir documentos completos9.
- Proporciona **soporte para transacciones distribuidas** y garantiza la entrega de mensajes en entornos empresariales complejos

Servicios SOAP



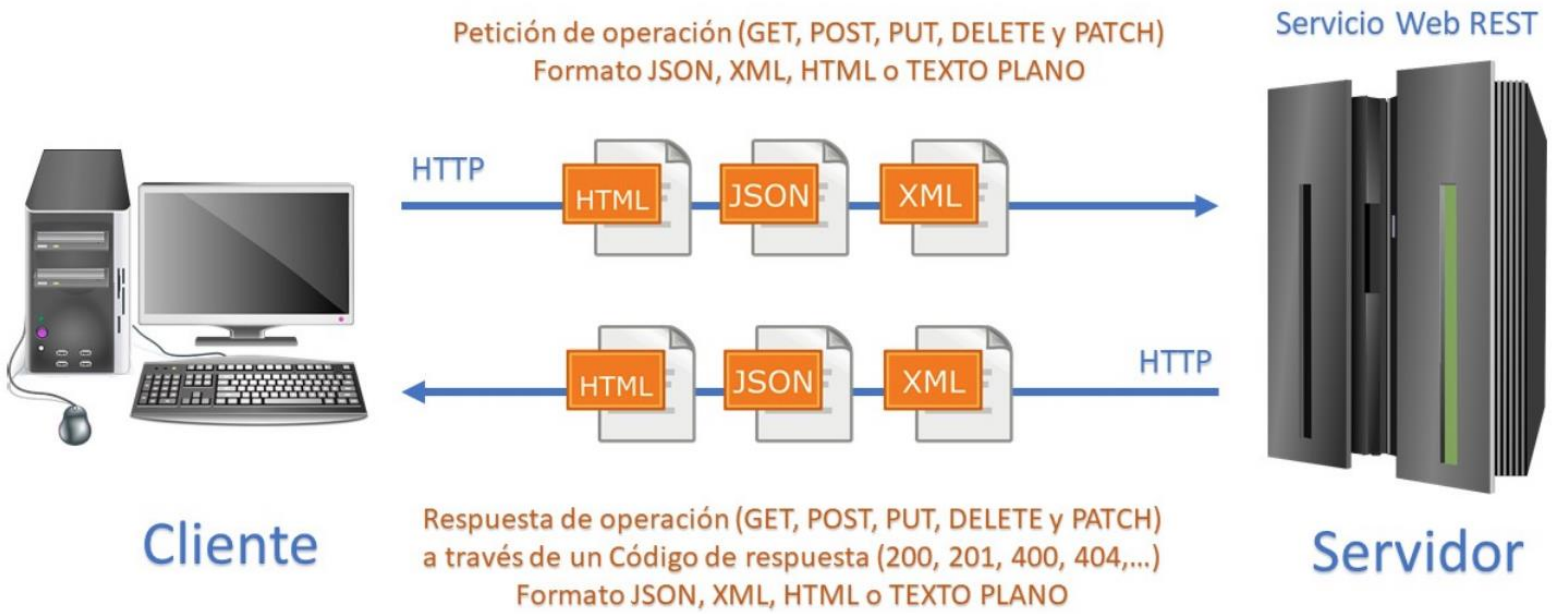
Servicios REST

- Un servicio **REST (Representational State Transfer)** tiene las siguientes características principales:
- **Arquitectura Cliente-Servidor**: REST separa el cliente (quien consume los recursos) del servidor (quien los proporciona). Esto permite que ambos evolucionen de manera independiente, mejorando la escalabilidad y flexibilidad del sistema.
- Sin Estado (**Stateless**): Cada solicitud del cliente al servidor debe contener toda la información necesaria para ser procesada. El servidor no guarda información de estado entre solicitudes, lo que facilita la escalabilidad y simplifica el diseño.
- En REST, **todo se considera un recurso**, como usuarios, productos o datos. Cada recurso es identificado de manera única mediante una URI (Uniform Resource Identifier), lo que facilita el acceso y manipulación de los mismos.
- REST utiliza **métodos HTTP** como:
  - **GET**: Recuperar información.
  - **POST**: Crear un nuevo recurso.
  - **PUT**: Actualizar un recurso existente.
  - **DELETE**: Eliminar un recurso.

Servicios REST

- Los recursos pueden representarse en diferentes formatos, como **JSON, XML, HTML o texto plano**. JSON es el formato más utilizado debido a su ligereza y facilidad de uso en aplicaciones web y móviles.
- Las respuestas del servidor pueden ser etiquetadas como **cacheables o no cacheables**. Esto permite almacenar respuestas en caché para mejorar el rendimiento y reducir la carga en el servidor.
- REST define una **interfaz uniforme** para interactuar con los recursos. Esto incluye:
  - Uso consistente de URIs.
  - Mensajes autodescriptivos que contienen toda la información necesaria para procesar la solicitud.
  - Separación clara entre las operaciones realizadas y los datos manipulados.
- Uso de **HATEOAS** (Hypermedia as the Engine of Application State): Las respuestas del servidor pueden incluir enlaces a otros recursos relacionados, permitiendo al cliente navegar por la API sin necesidad de conocer previamente su estructura completa.
- **Sistema en Capas**: REST permite la inclusión de capas intermedias (como servidores proxy o gateways) para mejorar la seguridad, escalabilidad y rendimiento del sistema.

Servicios REST



Diferencias entre servicios SOAP y servicios REST

Característica	SOAP	REST
Arquitectura	Protocolo rígido	Estilo arquitectónico flexible
Formato de datos	Solo XML	JSON, XML, HTML, texto plano
Estado	Con o sin estado	Sin estado
Rendimiento	Más lento	Más rápido
Seguridad	WS-Security	HTTPS (básico)
Casos de uso	Transacciones complejas	Aplicaciones móviles/web

Realización de pruebas web incluyendo métodos de captura y repetición de pruebas

Tema 1

Pruebas funcionales, de carga y regresión

- Las pruebas funcionales, de carga y de regresión son fundamentales para garantizar la calidad y el rendimiento de los servicios web.
    - **Pruebas Funcionales**: Estas pruebas verifican que el servicio web cumple con los requisitos funcionales especificados. Se centran en comprobar que cada función o característica del sistema funciona correctamente según lo esperado.
    - **Pruebas de carga**: Estas pruebas evalúan cómo se comporta un servicio web bajo diferentes niveles de carga, simulando usuarios concurrentes o solicitudes simultáneas. El objetivo es identificar cuántas solicitudes puede manejar antes de que su rendimiento se degrade.
    - **Pruebas de regresión**: Estas pruebas garantizan que los cambios recientes en el código (nuevas funciones, correcciones o actualizaciones) no afecten negativamente las funcionalidades existentes.

Pruebas Funcionales

- **Características:**
  - Validan la lógica de negocio del servicio web.
  - Se basan en los requisitos del sistema y no en su implementación interna (pruebas de caja negra).
  - Incluyen casos como:
    - Verificar que un servicio devuelve los datos correctos para una solicitud válida.
    - Comprobar el manejo adecuado de errores para solicitudes inválidas.
  - Su objetivo es garantizar que el servicio funcione correctamente desde la perspectiva del usuario final.

- **Ejemplo en servicios web:**
  - Probar si un servicio SOAP devuelve el XML esperado al enviar una solicitud válida.
  - Verificar si una API REST responde con un código HTTP 200 y datos correctos al realizar una consulta.

Pruebas de carga

- **Características**:
  - Simulan escenarios reales de uso para medir el rendimiento.
  - Ayudan a identificar cuellos de botella, límites de capacidad y puntos críticos de falla.
  - Evalúan métricas como tiempos de respuesta, uso de CPU, memoria y ancho de banda.
  - Son esenciales para garantizar que el sistema pueda manejar picos de tráfico sin fallar.

- **Ejemplo en servicios web:**
  - Simular 500 usuarios concurrentes accediendo a una API REST para verificar si los tiempos de respuesta permanecen aceptables.
  - Probar un servicio SOAP con solicitudes masivas para determinar cuándo comienza a fallar.

Pruebas de regresión

- **Características**:
  - Vuelven a ejecutar casos de prueba previamente exitosos para verificar que las funcionalidades existentes siguen funcionando correctamente.
  - Detectan efectos secundarios no deseados tras modificaciones en el sistema.
  - Son ideales para entornos con actualizaciones frecuentes, como metodologías ágiles o DevOps.
  - Pueden ser automatizadas para ahorrar tiempo y recursos.

- **Ejemplo en servicios web:**
  - Probar nuevamente todos los endpoints de una API REST después de agregar un nuevo endpoint.
  - Verificar que un cambio en el esquema XML de un servicio SOAP no afecta las respuestas existentes.

Comparación entre pruebas funcionales, de carga y regresión

Tipo de Prueba	Objetivo Principal	Ejemplo en Servicios Web
Funcionales	Validar que las funcionalidades cumplen requisitos	Verificar respuestas correctas a solicitudes válidas.
De Carga	Evaluar rendimiento bajo diferentes niveles de carga	Simular 500 usuarios concurrentes accediendo a una API REST.
De Regresión	Garantizar que cambios no afectan funcionalidades	Reprobar endpoints tras agregar nuevas funciones.

Pasos a seguir

- SoapUI es una herramienta poderosa para realizar pruebas web, incluyendo métodos de captura y repetición de pruebas.

    1. **Realización de pruebas Web**: SoapUI permite realizar pruebas funcionales, de carga y de regresión en servicios web SOAP y REST.

    2. **Métodos de captura (Recording):** SoapUI incluye un monitor HTTP que permite capturar tráfico entre el cliente y el servidor, lo cual es útil para analizar y repetir interacciones

    3. **Métodos de repetición (Replay)**: SoapUI permite reutilizar las capturas realizadas mediante la función de repetición, ideal para pruebas de regresión o escenarios controlados

1.- Realización de pruebas Web

- Los pasos necesarios serán:
  - Crear un proyecto
  - Configurar un TestSuite
  - Definir TestCases
  - Agregar las afirmaciones o aserciones

2.- Métodos de captura (Recording)

- Para poder capturar el tráfico HTTP seguiremos estos pasos:
  - Iniciar el monitor HTTP
  - Capturar peticiones/respuestas
  - Guardar capturas como TestCase

3.- Métodos de repetición (Replay)

- Para repetir pruebas seguiremos estos pasos:
  - Modificar parámetros en cabeceras y cuerpo de la petición
  - Ejecutar el TestCase
  - Usar Mock Services
  - Ejecutar pruebas automatizadas

Ventajas del enfoque captura - repetición

- Permite analizar tráfico real entre cliente y servidor.
- Facilita la creación rápida de pruebas automatizadas basadas en escenarios reales.
- Ayuda a realizar pruebas regresivas sin necesidad de interactuar con sistemas externos constantemente.
- Simula entornos controlados mediante Mock Services

Monitorización y control de acciones web

Tema 2

Monitorización y control de acciones web

- Para llevar a cabo la monitorización y control de acciones web en SoapUI, puedes utilizar varias funcionalidades que permiten supervisar el rendimiento y la disponibilidad de tus APIs.
    - Monitorización de APIs con AlertSite
    - Grabación y Análisis de Tráfico HTTP con HTTP Monitor

- La monitorización y control de acciones web en SoapUI mediante AlertSite y el HTTP Monitor te permite:
    - Realizar un seguimiento continuo del rendimiento y disponibilidad de tus APIs.
    - Capturar tráfico HTTP para análisis detallado, facilitando la identificación de problemas.
    - Reutilizar las interacciones grabadas para pruebas futuras o simulaciones con MockServices.

Grabación de scripts

Tema 3

Grabación de Scripts

- En SoapUI, no existe una funcionalidad automática para "grabar" scripts como en algunas herramientas de automatización de pruebas (por ejemplo, Selenium IDE). Sin embargo, puedes escribir y guardar manualmente tus scripts Groovy para reutilizarlos en diferentes proyectos o casos de prueba.
    - *Con Groovy Script Test Step* dentro de SoapUI, puedes guardar el script como parte del proyecto. Esto significa que el script se almacenará dentro del archivo del proyecto SoapUI (*.xml*) y estará disponible siempre que abras el proyecto.
    - SoapUI permite guardar scripts Groovy externos en una carpeta específica para que puedan ser reutilizados en diferentes proyectos.
    - Puedes diseñar tus scripts para que acepten parámetros dinámicos y sean más reutilizables.
- Con estas opciones, puedes guardar, organizar y reutilizar tus scripts Groovy de manera eficiente en SoapUI. La biblioteca de scripts es especialmente útil si trabajas con múltiples proyectos o necesitas estandarizar tus pruebas automatizadas.

Grabación de estadísticas mientras se ejecuta una prueba

Tema 4

Grabación de estadísticas

- En SoapUI, puedes grabar estadísticas mientras ejecutas una prueba, especialmente durante pruebas de carga (*Load Tests*), para analizar el rendimiento y comportamiento del sistema bajo prueba.
- SoapUI permite registrar estadísticas en tiempo real y exportarlas para análisis posterior. Esto se configura principalmente en las opciones de *LoadTest*. Los pasos a seguir:
    1. Crear un LoadTest
    2. Habilitar el registro de estadísticas
    3. Ejecutarlo
- Cuando ejecutes el LoadTest, SoapUI generará archivos CSV en la carpeta especificada. Cada archivo contendrá estadísticas específicas para cada paso de prueba (TestStep) y un resumen general.
    1. Exportar datos manualmente
    2. Validación y registro de errores
    3. Funciones avanzadas como:
        - Monitoreo en tiempo real de recursos del servidor (CPU, memoria, etc.).
        - Distribución de pruebas en múltiples máquinas locales o en la nube.
        - Generación automática de informes detallados con gráficos y métricas clave.

Configuración: Project, TestSuite,TestCase, Steps

Tema 5

Configuración: Project, TestSuite, TestCase y TestStep

- En SoapUI, la estructura básica de configuración para automatizar pruebas de API se organiza en una jerarquía: Project > TestSuite > TestCase > TestStep.
  - El **Project** es el nivel más alto en la jerarquía y representa tu conjunto de pruebas para una API o servicio.
  - Un **TestSuite** agrupa varios casos de prueba (TestCases) relacionados. Es útil para organizar pruebas por funcionalidades o módulos.
  - Un **TestCase** agrupa pasos específicos (TestSteps) que ejecutan una prueba completa, como enviar solicitudes, validar respuestas o ejecutar scripts.
  - Un **TestStep** es una acción individual dentro de un caso de prueba (TestCase). Puede ser una solicitud HTTP/SOAP, una validación, un script Groovy, entre otros.

Nivel	Descripción	Ejemplo
Project	Representa la API o servicio a probar	`API Usuarios`
TestSuite	Agrupa casos de prueba relacionados	`Gestión de Usuarios`
TestCase	Define los pasos necesarios para una prueba	`Crear Usuario`
TestStep	Acciones individuales dentro del caso de prueba	*REST Request*, *Groovy Script*

Testing manual con SOAP: Assertions , Propiedades y Property Transfer. Operaciones con request y response

Tema 6

Uso de Assertions para Validar Respuestas

- Las assertions son puntos de validación que verifican si la respuesta cumple con los criterios esperados. Se aplican a nivel de TestStep. Los tipos mas comunes:

1. **Contains/Not Contains:**

    - Valida si el texto esperado está presente/ausente en la respuesta.
    - Ejemplo para validar un mensaje de éxito:

```
<soap:Envelope>
    <soap:Body>
        <m:ConversionRateResponse>
            <m:ConversionRateResult>1.2</m:ConversionRateResult>
        </m:ConversionRateResponse>
    </soap:Body>
</soap:Envelope>
```

    - Assertion: Contains con valor 1.2

Uso de Assertions para Validar Respuestas

2. XPath Match:

- Extrae valores específicos usando expresiones XPath.
- Ejemplo para validar un tipo de token:

```groovy
// XPath: //*:token_type/text()
Assert Expected Value: "online"
```

- Si la respuesta incluye <token_type>online</token_type>, la assertion pasa

Uso de Assertions para Validar Respuestas

3. **Response SLA:**

    - Verifica que el tiempo de respuesta no supere un límite (ej: 500 ms).

- Cómo agregar una assertion?

    - Haz clic en el TestStep > Pestaña Assertions > + > Selecciona el tipo.
    - Configura la condición (ej: XPath, texto esperado)

Propiedades

- Las propiedades almacenan valores reutilizables en distintos niveles (proyecto, TestSuite, TestCase).
- Usar propiedades en requests: En el cuerpo de una solicitud SOAP/REST:

```
<soap:Envelope>
    <soap:Body>
        <m:GetUser>
            <m:Endpoint>${#Project#BaseURL}/users</m:Endpoint>
        </m:GetUser>
    </soap:Body>
</soap:Envelope>
```

Transferencia de datos (Property Transfer)

- Transfiere valores entre respuestas y solicitudes usando el *Property Transfer TestStep.*

- **Ejemplo:**

  - Supongamos que tienes un servicio web que permite iniciar sesión y devuelve un access_token en la respuesta. Quieres usar este token en solicitudes posteriores para acceder a recursos protegidos.

Operaciones con request y response

- Las pruebas de rendimiento son un tipo de prueba no funcional que evalúa cómo un servicio web responde bajo diferentes condiciones de carga. Su objetivo principal es identificar problemas relacionados con la velocidad, estabilidad, escalabilidad y capacidad del sistema.

Testing manual con REST

Tema 7

Testing manual con REST

- Para realizar testing manual con REST en SoapUI, estos son los pasos a seguir:

    1. Crear un proyecto REST
    2. Configurar una solicitud REST: Agregar TestSuite, TestCase y TestStep
    3. Validar la respuesta con assertions
    4. Uso de propiedades para almacenar valores dinámicos
    5. Realizar operaciones CRUD con GET, POST, PUT, DELETE
    6. Análisis manual:
        - Los códigos de estado HTTP devueltos por cada operación (200, 201, etc.).
        - La consistencia de los datos entre las operaciones (GET, PUT, etc.).
        - Los tiempos de respuesta para garantizar que cumplen con los criterios esperados.

Pruebas síncronas y asíncronas

Tema 8

Pruebas síncronas y asincronas

- En SoapUI, puedes realizar pruebas tanto síncronas como asíncronas para servicios web.
    - Las pruebas **síncronas** implican que el cliente envía una solicitud y espera inmediatamente una respuesta del servicio. Este es el caso más común en servicios SOAP.
    - Las pruebas **asíncronas** son más complejas, ya que implican que el cliente envía una solicitud y no recibe una respuesta inmediata. En cambio, el servidor puede enviar una respuesta más tarde o invocar un callback en el cliente

Automatización de pruebas (ant, maven, jenkins)

Tema 9

Automatización de pruebas (ant, maven, jenkins)

- SoapUI puede integrarse con herramientas como Ant, Maven y Jenkins para automatizar pruebas de servicios web.

1. Automatización con **Ant**:

    - Aunque SoapUI no tiene una tarea específica para Ant, puedes usar el comando SoapUITestCaseRunner en un script de Ant para ejecutar pruebas.

2. Automatización con **Maven**:

    - SoapUI tiene un plugin oficial para Maven que permite ejecutar pruebas como parte del ciclo de vida de construcción.

```
<plugin>
    <groupId>com.smartbear.soapui</groupId>
    <artifactId>soapui-maven-plugin</artifactId>
    <version>5.6.0</version>
```

3. Automatización con **Jenkins**

    - Jenkins permite ejecutar proyectos SoapUI como parte de su flujo CI/CD, utilizando comandos o integraciones con Maven.

Ventajas del enfoque automatizado

- Ventajas:

  1. **Integración Continua**: Ejecuta pruebas automáticamente cada vez que se realiza un cambio en el código.
  2. **Escalabilidad:** Permite manejar grandes conjuntos de pruebas sin intervención manual.
  3. **Informes Detallados**: Genera reportes que facilitan el análisis y seguimiento de errores.

- Este enfoque asegura que tus servicios web sean probados exhaustivamente y estén listos para producción mediante herramientas modernas como Ant, Maven y Jenkins.

Servicios Mock: con SOAP Y REST

Tema 10

Servicios Mock

- SoapUI permite crear servicios Mock tanto para SOAP como para REST, lo que resulta útil para simular servicios web y realizar pruebas sin depender de un entorno real.

1. **Servicios Mock SOAP**

    - Un servicio Mock SOAP simula un servicio basado en WSDL. Puedes usarlo para:
        - Prototipar un servicio web.
        - Realizar pruebas funcionales y de carga antes de implementar el servicio real.
        - Simular respuestas específicas para probar clientes.

2. **Servicios Mock REST**

    - Un servicio Mock REST simula una API RESTful. Puedes usarlo para:
        - Probar aplicaciones front-end mientras se desarrolla el back-end.
        - Simular diferentes escenarios de respuesta (éxito, error, etc.).
        - Prototipar endpoints REST.

Comparación entre servicios Mock SOAP y REST

Característica	SOAP Mock Service	REST Mock Service
Tipo de protocolo	Basado en WSDL/SOAP	Basado en HTTP/REST
Formato de respuesta	XML	JSON, XML o texto
Uso común	Simulación de servicios empresariales	Simulación de APIs modernas
Métodos soportados	Operaciones definidas por WSDL	GET, POST, PUT, DELETE
Configuración avanzada	Scripts Groovy para respuestas dinámicas	Scripts Groovy para rutas y contenido

Groovy básico

Tema 11

Sintaxis básica de Groovy

1. Variables

- Groovy utiliza tipado dinámico, lo que permite declarar variables sin especificar su tipo explícitamente. Usa la palabra clave def para definir variables.

```
def x = 42  // Integer
def y = "Hola Mundo"  // String
def z = true  // Boolean
```

- Las variables pueden cambiar de tipo dinámicamente:

```
x = "Nuevo valor"
println x.getClass()  // Muestra el tipo actual de la variable
```

Sintaxis básica de Groovy

2. Strings

- Comillas simples (''): Para cadenas simples sin interpolación.
- Comillas dobles (""): Permiten interpolación de variables (GString).
- Comillas triples (''' '''): Para cadenas multilínea.

```
def simpleString = 'Texto simple'
def interpolatedString = "Hola, ${name}!"
def multiLineString = '''Línea 1
Línea 2
Línea 3'''
```

- Puedes concatenar cadenas usando +, ${} o el operador <<:

```
def name = "Juan"
println "Hola, " + name + "!"
println "Hola, ${name}!"
def sb = new StringBuilder()
sb << "Hola, " << name << "!"
println sb.toString()
```

3. Literales

- Groovy admite literales para representar valores fijos:
  - Enteros: 12
  - Decimales: 1.45
  - Caracteres: 'a'
  - Cadenas: "Texto" o 'Texto'

4. Operadores

- Groovy soporta operadores aritméticos, relacionales y lógicos:

```
println 5 + 3    // Suma
println 10 > 5   // Relacional
println true && false   // Lógico
```

- También permite sobrecarga de operadores para colecciones y cadenas:

```
def lista = [1, 2, 3]
lista << 4   // Agregar elemento a la lista
println lista
```

5. Estructuras de control

- Condicionales:

```
if (x > 10) {
    println "Mayor que 10"
} else {
    println "Menor o igual a 10"
}
```

- Bucles:

```
5.times { println "Iteración $it" }
(1..5).each { println it }
```

6. Métodos

- Los métodos se declaran con la palabra clave def y pueden devolver valores opcionales:

```
def suma(a, b) {
    return a + b
}
println suma(5, 3)
```

7. Clases

- Groovy admite clases similares a Java pero con menos sintaxis obligatoria:

```groovy
class Persona {
    String nombre

    def saludar() {
        println "Hola, mi nombre es ${nombre}"
    }
}

def persona = new Persona(nombre: "Juan")
persona.saludar()
```

8. Importaciones

- Groovy incluye algunas bibliotecas por defecto, pero puedes importar otras según sea necesario:

```
import groovy.xml.MarkupBuilder

def xml = new MarkupBuilder()
```

9. Substring

- Puedes extraer partes de cadenas usando índices o rangos:

```
def texto = "Hola Mundo"
println texto[0..3]   // Hola
println texto[-5..-1] // Mundo
```

10. Assertions

- Las assertions son útiles para validar expresiones en Groovy:

```
assert 2 + 2 == 4   // Pasa sin errores si la condición es verdadera.
```

Informes con SoapUI PRO

Tema 12

Informes con SoapUI

- Para realizar un informe seguiremos estos pasos:

    1. Crear el proyecto
    2. Configurar TestSuites y TestCases
    3. Ejecutar las pruebas
    4. Generar los informes
    5. Personalizar el informe si fuese necesario

Pruebas de rendimiento: Creación y análisis de resultados

Tema 13

Pruebas de Rendimiento

- Las pruebas de rendimiento son un tipo de prueba no funcional que evalúa cómo un servicio web responde bajo diferentes condiciones de carga. Su objetivo principal es identificar problemas relacionados con la velocidad, estabilidad, escalabilidad y capacidad del sistema.
- Las pruebas de rendimiento son esenciales para garantizar que los servicios web sean rápidos, estables y escalables, ofreciendo una experiencia óptima a los usuarios incluso bajo condiciones exigentes
- Realizar pruebas de rendimiento ofrece múltiples beneficios que son esenciales para garantizar la calidad y eficiencia de aplicaciones y servicios web, por ejemplo:
    - Identificación de Cuellos de Botella
    - Mejora de la Experiencia del Usuario
    - Aumento de la Disponibilidad del Servicio
    - Optimización de Recursos

Características de las Pruebas de Rendimiento

1. **Evaluación de la capacidad del sistema**:
   - Determinan cuántos usuarios o solicitudes simultáneas puede manejar el servicio web sin degradar su rendimiento.
2. **Medición de tiempos clave**:
   - Analizan el tiempo de respuesta, el tiempo de procesamiento y la velocidad de transferencia de datos.
3. **Identificación de cuellos de botella**:
   - Detectan problemas como uso excesivo de CPU, memoria, red o disco.
4. **Pruebas bajo diferentes condiciones**:
   - Simulan cargas normales, altas o inesperadas para evaluar la respuesta del sistema.
5. **Validación de escalabilidad**:
   - Verifican si el sistema puede manejar un aumento en el tráfico ajustando los recursos disponibles.

Tipos principales de Pruebas de Rendimiento

1. **Pruebas de Carga:**
   - Evalúan el rendimiento del servicio bajo una carga específica (por ejemplo, 1000 usuarios concurrentes).
   - Objetivo: Garantizar que el sistema funcione correctamente bajo condiciones normales.
2. **Pruebas de Estrés:**
   - Llevan al sistema más allá de su capacidad máxima para identificar su punto de quiebre.
   - Objetivo: Evaluar cómo se comporta el sistema ante condiciones extremas y su capacidad para recuperarse.
3. **Pruebas de Resistencia (Soak Testing):**
   - Evalúan la estabilidad del sistema sometiéndolo a una carga constante durante un período prolongado.
   - Objetivo: Detectar problemas como fugas de memoria o degradación del rendimiento a largo plazo.
4. **Pruebas de Picos (Spike Testing)**:
   - Simulan aumentos repentinos e inesperados en la carga para evaluar cómo responde el sistema.
   - Objetivo: Verificar si el servicio puede manejar picos repentinos sin fallar.
5. **Pruebas de Tiempo de Respuesta**:
   - Miden cuánto tiempo tarda el sistema en responder a solicitudes individuales bajo diferentes niveles de carga.
   - Objetivo: Garantizar que los tiempos cumplan con los requisitos establecidos.

Security Test: Creación y análisis de resultados

Tema 14

Pruebas de seguridad

- Las pruebas de seguridad en SoapUI permiten identificar vulnerabilidades comunes en APIs y servicios web, como:
  - **SQL Injection**: Inyección de consultas SQL maliciosas.
  - **XPath Injection**: Manipulación de consultas XML.
  - **Fuzzing**: Envío de datos aleatorios para probar la robustez del servicio.
  - **XML Bombs**: Ataques que explotan la estructura XML para sobrecargar el sistema.
  - **Exposición de archivos sensibles**: Verificación de acceso no autorizado a recursos.

- Estas pruebas son críticas para garantizar que tus servicios sean seguros y resistentes frente a ataques.
- Las pruebas de seguridad en SoapUI son esenciales para garantizar que tus servicios web sean resistentes frente a ataques comunes como inyección SQL, manipulación XML y exposición a datos sensibles. Con herramientas como *Security Scans*, logs detallados e informes automatizados, puedes identificar y mitigar riesgos antes de implementar tus APIs en producción.

Icono *Training Consulting*

SOAP UI para verificar Web Services

Completa nuestra encuesta de satisfacción a través del QR

GRACIAS

indra

2025