DevOps

INTRODUCCIÓN

© JMA 2020. All rights reserved

Introducción

- El término DevOps como tal se popularizó en 2009, a partir de los “DevOps Days” celebrados primero en Gante (Bélgica) y está fuertemente ligado desde su origen a las metodologías ágiles de desarrollo software.
- DevOps está destinado a denotar una estrecha colaboración entre lo que antes eran funciones puramente de desarrollo, funciones puramente operativas y funciones puramente de control de calidad.
- Debido a que el software debe lanzarse a un ritmo cada vez mayor, el antiguo ciclo de desarrollo-prueba-lanzamiento de "cascada" se considera roto. Los desarrolladores también deben asumir la responsabilidad de la calidad de los entornos de prueba y lanzamiento.

Introducción

- DevOps establece una “intersección” entre Desarrollo, Operaciones y Calidad, pero no se rige por un marco estándar de prácticas, es una cultura o movimiento que permite una interpretación mucho más flexible en la medida en que cada organización quiera llevarlo a la práctica, según su estructura y circunstancias.
- El objetivo final de DevOps es minimizar el riesgo de los cambios que se producen en las entregas y dar así un mayor valor tanto a los clientes como al propio negocio.
- La característica clave de DevOps es derribar las tradicionales barreras entre los desarrolladores, testers y especialistas en operaciones (Sistemas / Infraestructura) para que trabajen en colaboración a través de herramientas compartidas, corrigiendo diferencias entre personas y entre objetivos, creando vínculos más cercanos entre los participantes con objetivos en común. Además, incorporar el feedback de los clientes/usuarios en el proceso de desarrollo para acelerar la respuesta a errores y mejoras.

Principios

- Según DASA™ (DevOps Agile Skills Association) DevOps presenta 6 principios necesarios para la entrega de servicios TIC.
- Estos principios son:
  1. Acción centrada en el cliente (valor de actuación, innovación)
  2. Crear con el fin en mente (pensamiento de producto y servicio, mentalidad de ingeniero, colaboración)
  3. Responsabilidad Extremo a Extremo (gestión de gastos, nicho, soporte de rendimiento)
  4. Equipos autónomos interfuncionales (perfiles TI, habilidades complementarias)
  5. Mejora continua (fallos rápidos, experimentos)
  6. Automatizar todo lo que se pueda (mejora de la calidad, maximizar el flujo)

Principios

- Acción centrada en el cliente, que se manifiesta por medio del coraje para exponer las disfunciones y liderar cambios; y en la cultura abierta de innovación necesaria para realizar los cambios. Es imprescindible maximizar el retorno de la inversión existente en software, a la vez que innovar y adoptar nuevas tecnologías.
- Crear teniendo en cuenta el objetivo final, lo que requiere una visión amplia de producto y servicio, y no sólo de proyecto, una colaboración entre todas las partes implicadas desde la concepción hasta la operación.
- Responsabilidad extremo a extremo (end to end), hace que todos los participantes se sientan igualmente responsables del producto en todas su etapas, en lugar de poner foco en completar su parte y olvidarse de los demás. A efectos prácticos esto se traduce en trabajo en equipo, combinar el desarrollo y las operaciones para crear un equipo unilateral que se enfoque en la entrega de objetivos comunes.

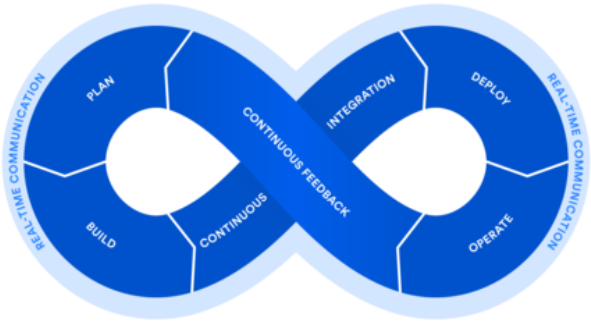
Principios

- Equipos autónomos multidisciplinares (cross funtional), no es posible tener la visión y responsabilidad extremos a extremo si no se cuenta con equipos capacitados para hacerse cargo del ciclo de vida completo del producto. Además, estos equipos necesitan tener tanto la competencia como la autonomía necesaria para hacerse realmente responsables de su trabajo.
- Mejora continua. El entorno abierto de innovación mencionado anteriormente requiere un contexto, y sobre todo una actitud que fomente la mejora continua, y el inconformismo. Mejorar requiere experimentar, no tener miedo al fracaso sino aprender de él y medir, que es la única forma de evaluar si se produce o no la mejora buscada.
- Automatizar todo lo que se pueda. La aproximación que promueve DevOps, requiere dedicar los recursos disponibles a las actividades que ofrezcan más valor, dejando de lado las que no lo aporten o automatizando todo lo posible. De esa manera, las personas se centran en lo verdaderamente importante y valioso para el objetivo de la organización.

Retos culturales

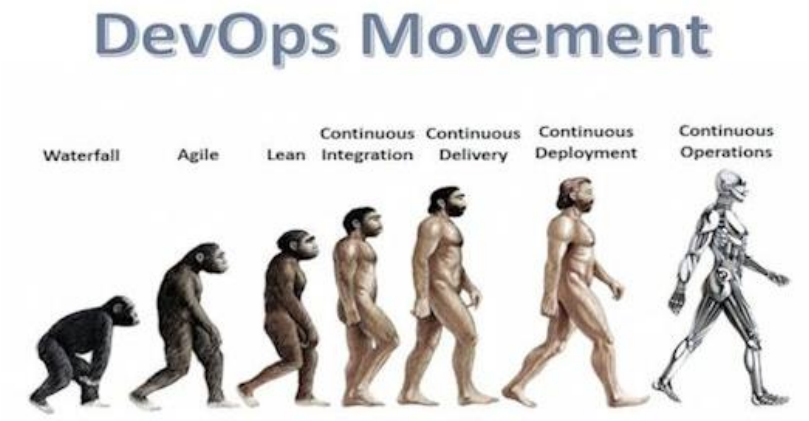
- Para poder realizar el cambio cultural que supone la implementación del modelo DevOps la organización tiene que ser capaz de trabajar en los siguientes factores:
  - Aprendizaje continuo:
    - Los equipos se auto-organizan, facilitan sesiones de aprendizaje, …
  - Experimentación:
    - Ofrecer tiempo, entornos aislados (sandbox), eliminar barreras, fallo controlado
  - Calidad en cada versión del producto o servicio:
    - Responsabilidad del producto entregado, automatización
  - Cultura de ingeniero:
    - Ambiciones compartidas por el equipo, experimentación
  - Cultura de efectividad:
    - El fin en mente, retrospectivas, filosofía Lean
  - Cultura de pensamiento en el producto o servicio:
    - Feedback de usuario, story boards
  - Cultura de toma de responsabilidades:
    - No explicar ‘cómo’, explicar ‘qué’, transparencia, pensamiento centrado en el cliente

Ciclo de vida



- Plan: identificar qué funcionalidad se quiere resolver
- Construir: el desarrollo puro, escribir código, pruebas unitarias y documentar lo que se vea necesario
- Integración Continua: automatizar desde el código hasta el entorno de producción
- Desplegar: automatizar el paso a producción
- Operar: vigilar el correcto funcionamiento del entorno, monitorizar
- Feed back: verificar que la funcionalidad tiene valor para el usuario o el retorno esperado

Movimiento DevOps

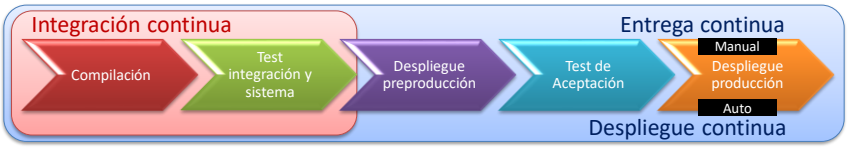


Ciclo “continuo”

- Muchas empresas se encuentran en algún lugar entre la cascada y las metodologías ágiles. DevOps realmente comienza donde está el “Lean” en la imagen. A medida que se eliminan los cuellos de botella y se comienza a brindar coherencia, podemos comenzar a avanzar hacia la integración continua, la entrega continua y, tal vez, hasta la implementación y las operaciones continuas.
- Con la integración continua, los desarrolladores necesitan escribir pruebas unitarias y se crea un proceso de construcción automatizado. Cada vez que un desarrollador verifica el código, se ejecuta automáticamente una prueba unitaria y, si alguna de ellas falla, la compilación completa falla. Esta es una mejora con respecto al modelo anterior porque se introducen menos errores en el control de calidad y la compilación solo contiene código de trabajo que reduce la acumulación de defectos.
- Al automatizar las pruebas unitarias en los procesos de construcción, eliminamos muchos errores humanos, mejorando así la velocidad y confiabilidad.

Ciclo “continuo”

- La entrega continua CD (continuous delivery) hace referencia a entregar las actualizaciones a los usuarios según estén disponibles sobre una base sólida y constante.
- La despliegue continuo CD (continuous deployment) es la automatización del despliegue de la entrega continua, que no exista intervención humana a la hora de realizar el despliegue en producción.
- Llegar a CI y CD son el objetivo que muchas empresas se esfuerzan por cumplir y están aprovechando el DevOps para ayudarles a lograrlo. Sin embargo, algunas empresas deben ir más allá porque pueden realizar entregas muchas veces al día o tener una presencia web muy popular.
- La implementación continua con solo presionar un botón pasa por la compilación, las pruebas automatizadas, la automatización de los entornos y la producción. Esto se vuelve importante con respecto a las operaciones continuas.



Automatización de la Prueba

- La automatización de la prueba permite ejecutar muchos casos de prueba de forma consistente y repetida en las diferentes versiones del sistema sujeto a prueba (SSP) y/o entornos. Pero la automatización de pruebas es más que un mecanismo para ejecutar un juego de pruebas sin interacción humana. Implica un proceso de diseño de productos de prueba, entre los que se incluyen:
  - Software.
  - Documentación.
  - Casos de prueba.
  - Entornos de prueba
  - Datos de prueba
- La Solución de Automatización Pruebas (SAP) debe permitir:
  - Implementar casos de prueba automatizados.
  - Monitorizar y controlar la ejecución de pruebas automatizadas.
  - Interpretar, informar y registrar los resultados de pruebas automatizadas.

Objetivos de la automatización de pruebas

- Mejorar la eficiencia de la prueba.
- Aportar una cobertura de funciones más amplia.
- Reducir el coste total de la prueba.
- Realizar pruebas que los probadores manuales no pueden.
- Acortar el período de ejecución de la prueba.
- Aumentar la frecuencia de la prueba y reducir el tiempo necesario para los ciclos de prueba.

Ventajas de la automatización de pruebas

- Se pueden realizar más pruebas por compilación ("build").
- La posibilidad de crear pruebas que no se pueden realizar manualmente (pruebas en tiempo real, remotas, en paralelo).
- Las pruebas pueden ser más complejas.
- Las pruebas se ejecutan más rápido.
- Las pruebas están menos sujetas a errores del operador.
- Uso más eficaz y eficiente de los recursos de pruebas
- Información de retorno más rápida sobre la calidad del software.
- Mejora de la fiabilidad del sistema (por ejemplo, repetibilidad, consistencia).
- Mejora de la consistencia de las pruebas.

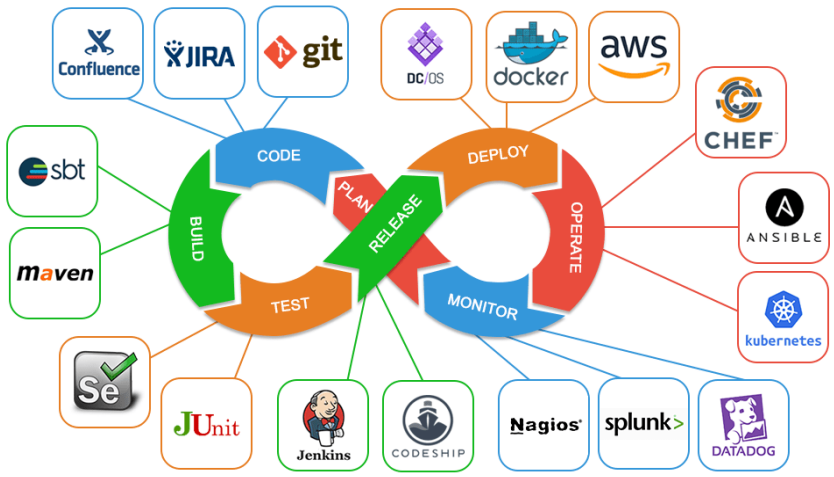
Desventajas de la automatización de pruebas

- Requiere tecnologías adicionales.
- Existencia de costes adicionales.
- Inversión inicial para el establecimiento de la SAP.
- Requiere un mantenimiento continuo de la SAP.
- El equipo necesita tener competencia en desarrollo y automatización.
- Puede distraer la atención respecto a los objetivos de la prueba, por ejemplo, centrándose en la automatización de casos de prueba a expensas del objetivo de las pruebas.
- Las pruebas pueden volverse más complejas.
- La automatización puede introducir errores adicionales.

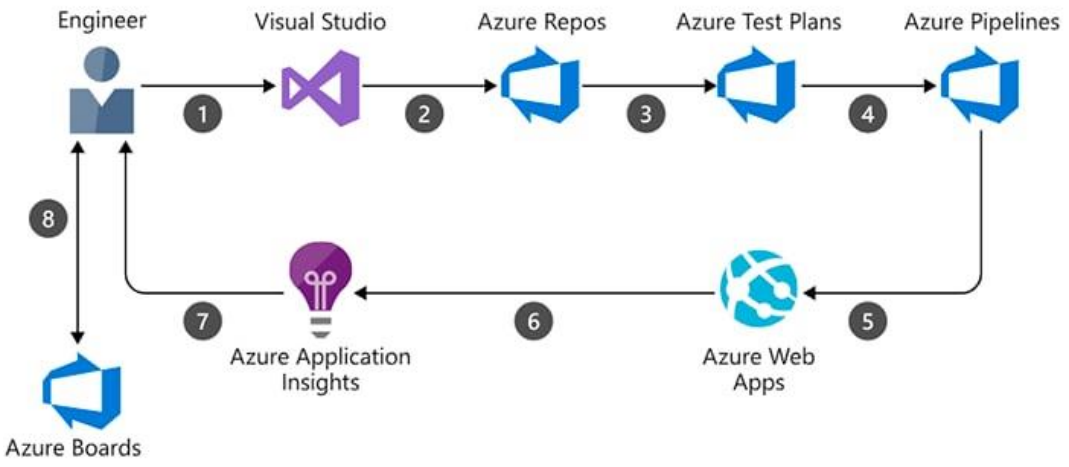
Limitaciones de la automatización de pruebas

- No todas las pruebas manuales se pueden automatizar.
- La automatización sólo puede comprobar resultados interpretables por la máquina.
- La automatización sólo puede comprobar los resultados reales que pueden ser verificados por un oráculo de prueba automatizado.
- No es un sustituto de las pruebas exploratorias.

Herramientas de Automatización



Azure DevOps



Azure DevOps

- Los puntos del esquema anterior exponen cómo sería un proceso de gestión del ciclo de vida de la una aplicación:
  1. Podemos usar nuestro Visual Studio para desarrollar nuestro código, aunque no es necesario usar esta herramienta en particular, podemos usar cualquier herramienta de desarrollo de código.
  2. De manera regular podremos ir guardando nuestro código en nuestro repositorio de Git dentro del módulo de Azure Repos.
  3. Mediante la integración continua, podemos desencadenar pruebas y la compilación de la aplicación mediante el módulo Azure Test Plans.
  4. Tras esto, con la implementación automatizada de los artefactos y la aplicación de valores de configuración específicos por entornos, podremos realizar despliegues a cada uno de los entornos que compongan nuestra infraestructura con los Azure Pipelines.
  5. En el caso de que no dispongamos de Azure Web Apps, los artefactos anteriormente mencionados también se pueden desplegar a nuestros servidores locales.
  6. Azure Application Insights es otra herramienta propia de Application Performance Management (APM) extensible para desarrolladores web en varias plataformas. Que pueden usar para supervisar y administrar la información sobre el estado, el rendimiento y el uso.
  7. Una vez analizados los resultados podemos volver al código.
  8. Mediante Azure Boards podremos hacer un seguimiento del trabajo diario y pendiente de los equipos

Prácticas de DevOps

- Más allá del establecimiento de una cultura de DevOps, los equipos ponen en práctica el método DevOps implementando determinadas prácticas a lo largo del ciclo de vida de las aplicaciones.
- Algunas de estas prácticas ayudan a agilizar, automatizar y mejorar una fase específica.
- Otras abarcan varias fases y ayudan a los equipos a crear procesos homogéneos que favorezcan la productividad.
- Estas practicas incluyen:
  - Integración y entrega continuas (CI/CD)
  - Control de versiones
  - Desarrollo ágil de software
  - Infraestructura como código
  - Administración de configuración
  - Supervisión continua

Control de versiones

- Control de versiones es la práctica de administrar el código por versiones, haciendo un seguimiento de las revisiones y del historial de cambios para facilitar la revisión y la recuperación del código.
- Esta práctica suele implementarse con sistemas de control de versiones, como Git, que permite que varios desarrolladores colaboren para crear código. Estos sistemas proporcionan un proceso claro para fusionar mediante combinación los cambios en el código que tienen lugar en los mismos archivos, controlar los conflictos y revertir los cambios a estados anteriores.
- El uso del control de versiones es una práctica de DevOps fundamental que ayuda a los equipos de desarrollo a trabajar juntos, dividir las tareas de programación entre los miembros del equipo y almacenar todo el código para poder recuperarlo fácilmente si fuese necesario.
- El control de versiones es también un elemento necesario en otras prácticas, como la integración continua y la infraestructura como código.

Desarrollo ágil de software

- El método ágil es un enfoque de desarrollo de software que hace hincapié en la colaboración en equipo, en los comentarios de los clientes y usuarios, y en una gran capacidad de adaptación a los cambios mediante ciclos cortos de lanzamiento de versiones.
- Los equipos que practican la metodología ágil proporcionan mejoras y cambios continuos a los clientes, recopilan sus comentarios y, después, aprenden y ajustan el software en función de lo que el cliente quiere y necesita.
- El método ágil es muy diferente a otros marcos más tradicionales, como el modelo en cascada, que incluye ciclos de lanzamiento de versiones largos definidos por fases secuenciales. Kanban y Scrum son dos marcos populares asociados al método ágil.

Infraestructura como código

- La infraestructura cómo código define las topologías y los recursos del sistema de un modo descriptivo que permite a los equipos administrar esos recursos igual que lo harían con el código. Las diferentes versiones de esas definiciones se pueden almacenar en sistemas de control de versiones, donde se pueden revisar y revertir, de nuevo, igual que el código.
- La práctica de la infraestructura como código permite a los equipos implementar recursos del sistema de un modo confiable, repetible y controlado. Además, la infraestructura como código ayuda a automatizar la implementación y reduce el riesgo de errores humanos, especialmente en entornos complejos de gran tamaño.
- Esta solución repetible y confiable para la implementación de entornos permite a los equipos mantener entornos de desarrollo y pruebas que sean idénticos al entorno de producción. De igual modo, la duplicación de entornos en otros centros de datos y en plataformas en la nube es más sencilla y más eficiente.

Administración de configuración

- Administración de la configuración hace referencia a la administración del estado de los recursos de un sistema, incluidos los servidores, las máquinas virtuales y las bases de datos.
- El uso de herramientas de administración de la configuración permite a los equipos distribuir cambios de un modo controlado y sistemático, lo que reduce el riesgo de modificar la configuración del sistema. Los equipos utilizan herramientas de administración de la configuración para hacer un seguimiento del estado del sistema y evitar alteraciones en la configuración, que es como se desvía la configuración de un recurso del sistema a lo largo del tiempo del estado definido para él.
- Al usarla junto con la infraestructura como código, resulta fácil elaborar plantillas y automatizar la definición y la configuración de sistemas, lo que permite a los equipos usar entornos complejos a escala.

Supervisión continua

- Supervisión continua significa tener visibilidad total y en tiempo real del rendimiento y el estado de toda la pila de aplicaciones, desde la infraestructura subyacente donde se ejecutan las aplicaciones hasta los componentes de software de niveles superiores.
- La visibilidad consiste en la recopilación de datos de telemetría y metadatos, así como en el establecimiento de alertas para condiciones predefinidas que garanticen la atención de un operador. La telemetría incluye registros y datos de eventos recopilados de varias partes del sistema que se almacenan donde pueden analizarse y consultarse.
- Los equipos de DevOps de alto rendimiento se aseguran de establecer alertas útiles que les permitan tomar medidas y recopilan datos de telemetría muy completos para obtener conclusiones a partir de enormes cantidades de datos.
- Estas conclusiones ayudan a los equipos a mitigar los problemas en tiempo real y a ver cómo mejorar la aplicación en futuros ciclos de desarrollo.

DevSecOps

DevSecOps

- DevSecOps significa desarrollo, seguridad y operaciones. Se trata de un enfoque que aborda la cultura, la automatización y el diseño de plataformas, e integra la seguridad como una responsabilidad compartida durante todo el ciclo de vida.
- DevOps no solo concierne a los equipos de desarrollo y operaciones. Si desea aprovechar al máximo la agilidad y la capacidad de respuesta de los enfoques de DevOps, la seguridad de la TI también debe desempeñar un papel integrado en el ciclo de vida completo de sus aplicaciones.
- Antes, el papel de la seguridad estaba aislado y a cargo de un equipo específico en la etapa final del desarrollo. Cuando los ciclos de desarrollo duraban meses o incluso años, no pasaba nada. Pero eso quedó en el pasado. Una metodología efectiva de DevOps garantiza ciclos de desarrollo rápidos y frecuentes (a veces de semanas o días), pero las prácticas de seguridad obsoletas pueden revertir incluso las iniciativas de DevOps más eficientes.

Desafíos

- **Los equipos son reacios a integrarse**: La esencia de DevSecOps es la integración de equipos para que puedan trabajar juntos en lugar de forma independiente. Sin embargo, no todo el mundo está preparado para hacer el cambio porque ya está acostumbrado a los procesos de desarrollo actuales.
- **Guerra de herramientas**: Dado que los equipos han estado trabajando por separado, han estado utilizando diferentes métricas y herramientas. En consecuencia, les resulta difícil ponerse de acuerdo sobre dónde tiene sentido integrar las herramientas y dónde no. No es fácil reunir herramientas de varios departamentos e integrarlas en una plataforma. Por lo tanto, el desafío es seleccionar las herramientas adecuadas e integrarlas adecuadamente para construir, implementar y probar el software de manera continua.
- **Implementar seguridad en IC/DC**: Generalmente, se ha pensado en la seguridad como algo que llega al final del ciclo de desarrollo. Sin embargo, con DevSecOps, la seguridad es parte de la integración y el desarrollo continuos (IC/DC). Para que DevSecOps tenga éxito, los equipos no pueden esperar que los procesos y herramientas de DevOps se adapten a los viejos métodos de seguridad. Al integrar los controles de seguridad en DevOps, las organizaciones están adoptando el nuevo modelo DevSecOps para aprovechar todo el potencial de IC/DC. Cuando las empresas implementan tecnologías de control de acceso o seguridad desde el principio, se aseguran de que esos controles estén en línea con un flujo de IC/DC.

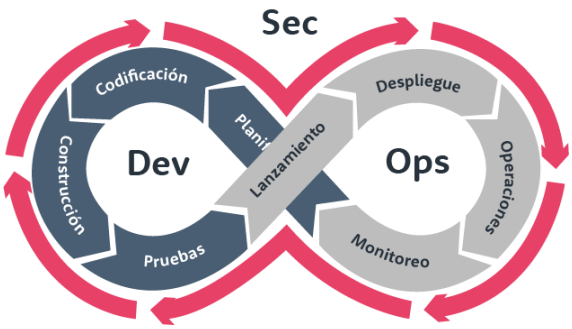
Beneficios

- Desarrollar software seguro desde el diseño.
- Identificar vulnerabilidades en el código y ataques antes.
- Aplicar la seguridad de forma más rápida y ágil.
- Responder a los cambios y requisitos rápidamente.
- Mejorar la colaboración y comunicación entre equipos.
- Generar una mayor conciencia sobre seguridad entre todos los miembros.
- Enfocarse en generar el mayor valor de negocio.

Implementación

- DevSecOps supone un cambio en la filosofía de trabajo, donde la seguridad se convierte en una responsabilidad compartida (extendida a los equipos de desarrollo y operaciones) e integrada a lo largo de todo el proceso de desarrollo y despliegue. Para poner en práctica DevSecOps, la seguridad debe convertirse en una condición más en el proceso de diseño, desarrollo y entrega de software.
- DevSecOps consta de dos partes diferenciadas pero que forman parte del mismo proceso.
  - Seguridad como código (SaC): cuando se incorporan la seguridad al desarrollo y las operación de herramientas informáticas o aplicaciones.
  - Infraestructura como código (IaC): hace referencia al conjunto de herramientas de desarrollo de operaciones (DevOps, por sus siglas en inglés) utilizadas para configurar y actualizar los componentes de la infraestructura para garantizar un entorno controlado. Trata las infraestructuras como un software.

Ciclo de vida



- Trabajar con el equipo de operaciones para llevar a cabo un mantenimiento de seguridad regular y actualizaciones de la infraestructura.
- Implementar pruebas de seguridad automatizadas para la arquitectura, diseño y despliegue en cada proyecto nuevo.
- Asesorar a los desarrolladores sobre las vulnerabilidades de seguridad para ayudar a configurar las herramientas y las opciones de diseño.
- Usar técnicas de monitorización continua para asegurar que los sistemas de seguridad están trabajando según lo previsto.

Buenas practicas

- **Implemente la automatización para proteger el entorno de IC/DC**: Uno de los aspectos clave del entorno IC/DC es la velocidad. Y eso significa que la automatización es necesaria para integrar la seguridad en este entorno, al igual que incorporar los controles y pruebas de seguridad esenciales a lo largo del ciclo de vida del desarrollo. También es importante implementar pruebas de seguridad automatizadas en las canalizaciones de IC/DC para permitir el análisis de vulnerabilidades en tiempo real.
- **Aborde los problemas de seguridad de la tecnología de código abierto**: Está aumentando el uso de herramientas de código abierto para el desarrollo de aplicaciones. Por lo tanto, las organizaciones deben abordar las preocupaciones de seguridad relacionadas con el uso de dichas tecnologías. Sin embargo, dado que los desarrolladores están demasiado ocupados para revisar el código fuente abierto, es importante implementar procesos automatizados para administrar el código fuente abierto, así como otras herramientas y tecnologías de terceros. Por ejemplo, utilidades como Open Web Application Security Project (OWASP) pueden verificar que no haya vulnerabilidades en el código que dependa de componentes de código abierto.
- **Integre el sistema de seguridad de la aplicación con el sistema de gestión de tareas**: Esto creará automáticamente una lista de tareas con errores que el equipo de seguridad de la información puede ejecutar. Además, proporcionará detalles procesables, incluida la naturaleza del defecto, su gravedad y la mitigación necesaria. Como tal, el equipo de seguridad puede solucionar problemas antes de que terminen en los entornos de desarrollo y producción.

Seguridad del entorno y de los datos

- **Estandarice y automatice el entorno**: los servicios deben tener la menor cantidad de privilegios posible para reducir las conexiones y los accesos no autorizados.
- **Centralice las funciones de control de acceso y de identidad de los usuarios**: el control de acceso estricto y los mecanismos de autenticación centralizados son fundamentales para proteger los microservicios, ya que la autenticación se inicia en varios puntos.
- **Aísle de la red y entre sí aquellos contenedores que ejecutan microservicios**: abarca tanto los datos en tránsito como en reposo, ya que ambos pueden ser objetivos valiosos para los atacantes.
- **Cifre los datos entre las aplicaciones y los servicios**: una plataforma de organización de contenedores con funciones de seguridad integradas disminuye las posibilidades de accesos no autorizados.
- **Incorpore puertas de enlace de API seguras**: las API seguras aumentan el control de los enrutamientos y las autorizaciones. Al disminuir la cantidad de API expuestas, las empresas pueden reducir las superficies de ataque.

Seguridad del proceso de CI/CD

- **Integre análisis de seguridad para los contenedores**: estos deberían ser parte del proceso para agregar contenedores al registro.
- **Automatice las pruebas de seguridad en el proceso de integración continua**: esto incluye ejecutar herramientas de análisis de seguridad estática como parte de las compilaciones, así como examinar las imágenes de contenedores prediseñadas para detectar puntos vulnerables de seguridad conocidos a medida que se incorporan al canal de diseño.
- **Incorpore pruebas automatizadas para las funciones de seguridad al proceso de pruebas de aceptación**: automatice las pruebas de validación de los datos de entrada, así como las funciones de verificación, autenticación y autorización.
- **Automatice las actualizaciones de seguridad, como la aplicación de parches para los puntos vulnerables conocidos**: lleve a cabo este proceso mediante el canal de DevOps. Esto elimina la necesidad de que los administradores inicien sesión en los sistemas de producción mientras crean un registro de cambios documentado y rastreable.
- **Automatice las funciones de gestión de la configuración de los sistemas y los servicios**: de esta manera, podrá cumplir con las políticas de seguridad y eliminar los errores manuales. También se recomienda automatizar los procesos de auditoría y corrección.

HERRAMIENTAS PARA PRUEBAS

Ecosistema

- Alrededor de las pruebas y el aseguramiento de la calidad existe todo un ecosistema de herramientas que se utilizan en una o más actividades de soporte de prueba, entre las que se encuentran:
  - Las herramientas que se utilizan directamente en las pruebas, como las herramientas de ejecución de pruebas, las herramientas de generación de datos de prueba y las herramientas de comparación de resultados.
  - Las herramientas que ayudan a gestionar el proceso de pruebas, como las que sirven para gestionar pruebas, resultados de pruebas, resultados de pruebas, datos requisitos, incidencias, defectos, etc., así como para elaborar informes y monitorizar la ejecución de pruebas.
  - Las herramientas que se utilizan en la fase de reconocimiento, como las herramientas de estrés y las herramientas que monitorizan y supervisan la actividad del sistema.
  - Cualquier otra herramienta que contribuya al proceso de pruebas sin ser especificas del mismo, como las hojas de cálculo, procesadores de texto, diagramadores, …
- Las herramientas pueden clasificarse en base a distintos criterios, tales como el objetivo, comercial/código abierto, especificas/integradas, tecnología utilizada …

Objetivos

- Mejorar la eficiencia de las tareas de pruebas automatizando tareas repetitivas o dando soporte a las actividades de pruebas manuales, como la planificación, el diseño, la elaboración de informes y la monitorización de pruebas.
- Automatizar aquellas actividades que requieren muchos recursos si se hacen de forma manuales (como por ejemplo, las pruebas estáticas, pruebas de GUI).
- Automatizar aquellas actividades que no pueden ejecutarse de forma manual (como por ejemplo , pruebas de rendimiento a gran escala de aplicaciones cliente-servidor).
- Aumentar la fiabilidad de las pruebas (por ejemplo, automatizando las comparaciones de grandes ficheros de datos y simulando comportamientos).

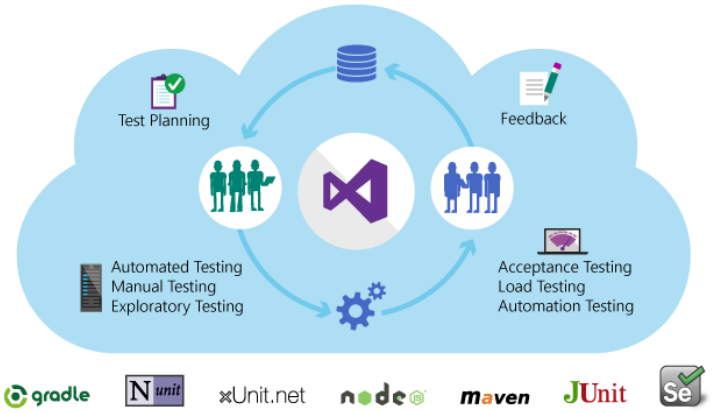
Ventajas

- Reducción del trabajo repetitivo (por ejemplo, la ejecución de pruebas de regresión, la reintroducción de los mismos datos de prueba y la comprobación contra estándares de codificación).
- Mayor consistencia y respetabilidad (por ejemplo las pruebas ejecutadas por una herramienta en el mismo orden y con la misma frecuencia, y pruebas derivadas de los requisitos).
- Evaluación de los objetivos (por ejemplo, medidas estáticas, cobertura).
- Facilidad de acceso a la información sobre las pruebas (por ejemplo, estadísticas y gráficos sobre el avance de las pruebas, la frecuencia de incidencias y el rendimiento).

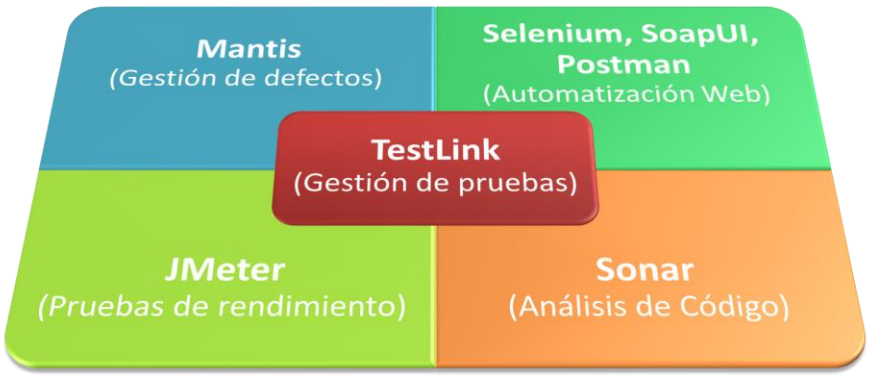
Desventajas

- Expectativas poco realistas de la herramienta (incluyendo funcionalidad y facilidad de uso).
- Exceso de confianza en la herramienta (sustitución por el diseño de pruebas o uso de pruebas automatizadas cuando seria mejor llevar a cabo pruebas manuales).
- Subestimar:
  - La cantidad de tiempo, coste y esfuerzo necesario para la introducción inicial de una herramienta (incluyendo formación y experiencia externa).
  - El tiempo y el esfuerzo necesarios para conseguir ventajas significativas y constantes de la herramienta (incluyendo la necesidad de cambios en el proceso de pruebas y la mejora continua de la forma en la que se utiliza la herramienta).
  - El esfuerzo necesario para mantener los activos de prueba generados por la herramienta.
- Desprecio del control de versión de los activos de prueba en la herramienta.
- Desprecio de problemas de relaciones e interoperabilidad entre herramientas críticas tales como las herramientas de gestión de requisitos, herramientas de control de versiones, herramientas de gestión de incidencias, herramientas de seguimiento de defectos y herramientas procedentes de varios fabricantes.
- Coste de las herramientas comerciales o ausencia de garantías en las herramientas open source.
- Riesgo de que el fabricante de la herramienta cierre, retire la herramienta o venda la herramienta a otro proveedor.
- Mala respuesta del fabricante para soporte, actualizaciones y corrección de defectos.
- Imprevistos, tales como la incapacidad de soportar una nueva plataforma.

Microsoft DevOps



Ecosistema de pruebas en Java



TestLink

- http://testlink.org/
- TestLink es una herramienta web de gestión de pruebas que ayuda a gestionar las pruebas funcionales de un proyecto permitiendo realizar las tareas relacionadas con el proceso de aseguramiento de calidad de software tales como: gestión de requerimientos, diseño de casos de prueba, planes de pruebas, ejecución de casos de prueba, seguimiento de informes de resultados del proceso de pruebas.
- Es una herramienta gratuita que permite crear y gestionar casos de pruebas y organizarlos en planes de prueba. Estos planes permiten a los miembros del equipo ejecutar casos de test y registrar los resultados dinámicamente, generar informes, mantener la trazabilidad con los requerimientos, así como priorizar y asignar tareas.
- Permite gestionar tantos proyectos de pruebas como sean necesarios, accesibles con diferentes roles de usuario con distintos permisos para los integrantes de un equipo de desarrollo.
- Los proyectos pueden definir diferentes plataformas de pruebas y mantener un inventario de las mismas.

Mantis

- https://www.mantisbt.org/
- Mantis Bug Tracker es un sistema de gestión de incidencias, control de cambios y control de errores, es una aplicación web OpenSource multiplataforma que permite gestionar las incidencias de la empresa, sistemas o proyectos.
- Es un sistema fácil de usar y adaptable a muchos escenarios, tanto para incidencias técnicas, peticiones de soporte o bugs de un sistema.
- Es una aplicación realizada con php y mysql que destaca por su facilidad y flexibilidad de instalar y configurar.
- Mantis es una aplicación que permite a distintos usuarios crear tickets de cualquier tipo.
- A la hora de notificar una incidencia, el usuario tiene muchas opciones y campos a rellenar con el fin de hacer más fácil el trabajo del encargado de resolver el ticket.
- Mantis permite notificar a los usuarios novedades por correo electrónico.
- Se puede especificar un número indeterminado de estados para cada tarea (nueva, se necesitan más datos, aceptada, confirmada, asignada, resuelta, cerrada) y configurar la transición de estados.
- Permite la carga de plugins específicos de la plataforma que añaden funcionalidades extra.

Selenium

- http://www.seleniumhq.org/
- El Selenium es un conjunto de herramientas para automatizar los navegadores web, robot que simula la interacción del usuario con el navegador, originalmente pensado como entorno de pruebas de software para aplicaciones basadas en la web.
- Como principales herramientas Selenium cuenta con:
  - Selenium IDE: una herramienta para grabar y reproducir secuencias de acciones con el navegador que permite crear pruebas sin usar un lenguaje de scripting para pruebas.
  - Selenium Core: API para escribir pruebas automatizadas y de regresión en un amplio número de lenguajes de programación populares incluyendo Java, C#, Ruby, Groovy, Perl, Php y Python.
  - WebDriver: interfaces que permite ejecutar las pruebas de forma nativa usando la mayoría de los navegadores web modernos en diferentes sistemas operativos como Windows, Linux y OSX.
  - Selenium Grid: Permite ejecutar muchas pruebas de un mismo grupo en paralelo o pruebas en múltiples entornos. Tiene la ventaja que un conjunto de pruebas muy grande puede dividirse en varias maquinas remotas para una ejecución más rápida o si se necesitan repetir las mismas pruebas en múltiples entornos.

Postman

https://www.getpostman.com/

- Postman surgió originariamente como una extensión para el navegador Google Chrome que permitía realizar peticiones API REST con métodos diferentes al GET. A día de hoy dispone de aplicaciones nativas para MAC, Windows y algunas producciones Linux.
- Está compuesto por diferentes herramientas y utilidades gratuitas (en la versión free) que permiten realizar tareas diferentes dentro del mundo API REST: creación de peticiones a APIs internas o de terceros, elaboración de tests para validar el comportamiento de APIs, posibilidad de crear entornos de trabajo diferentes (con variables globales y locales), y todo ello con la posibilidad de ser compartido con otros compañeros del equipo de manera gratuita (exportación de toda esta información mediante URL en formato JSON).
- Además, dispone de un modo cloud colaborativo (de pago) para que equipos de trabajo puedan desarrollar entre todos colecciones para APIs sincronizadas en la nube para una integración más inmediata y sincronizada.
- Quizás sea una de las herramientas más utilizadas para hacer testing exploratorio de API REST.

SoapUI

https://www.soapui.org

- SoapUI es una herramienta para probar servicios web que pueden ser servicios web SOAP, servicios web RESTful u otros servicios basados en HTTP.
- SoapUI es una herramienta de código abierto y completamente gratuita con una versión comercial, SoapUI Pro, que tiene una funcionalidad adicional para empresas con servicios web de misión crítica.
- SoapUI se considera el estándar de facto para las Pruebas de servicio API. Esto significa que hay mucho conocimiento en la red sobre la herramienta y blogs para obtener más información sobre el uso de SoapUI en la vida real.
- SoapUI permite realizar pruebas funcionales, pruebas de rendimiento, pruebas de interoperabilidad, pruebas de regresión y mucho más. Su objetivo es que la prueba sea bastante fácil de comenzar, por ejemplo, para crear una Prueba de carga, simplemente hay que hacer clic derecho en una prueba funcional y ejecutarla como una prueba de carga.
- SoapUI puede simular servicios web (mocking). Se puede grabar pruebas y usarlas más tarde.

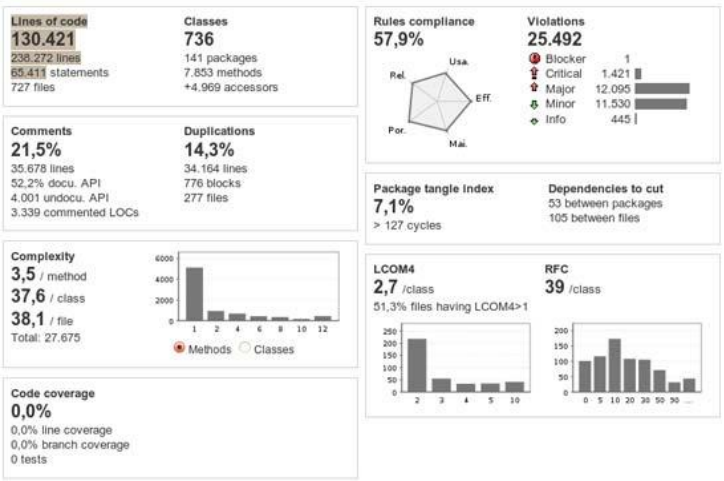
JMeter

- http://jmeter.apache.org
- JMeter es una herramienta de pruebas de carga para llevar acabo simulaciones sobre cualquier recurso de Software.
- Inicialmente diseñada para pruebas de estrés en aplicaciones web, hoy en día, su arquitectura ha evolucionado no sólo para llevar acabo pruebas en componentes habilitados en Internet (HTTP), sino también en Bases de Datos , programas en Perl, peticiones FTP y prácticamente cualquier otro medio.
- Además, posee la capacidad de realizar desde una solicitud sencilla hasta secuencias de peticiones que permiten diagnosticar el comportamiento de una aplicación en condiciones de producción.
- En este sentido, simula todas las funcionalidades de un navegador, o de cualquier otro cliente, siendo capaz de manipular resultados en determinada requisición y reutilizarlos para ser empleados en una nueva secuencia.

Sonar

- http://www.sonarqube.org/
- SonarQube (conocido anteriormente como Sonar) es una plataforma para la revisión y evaluación del código fuente.
- Es open source y realiza el análisis estático de código fuente integrando las mejores herramientas de medición de la calidad de código como Checkstyle, PMD o FindBugs, para obtener métricas que pueden ayudar a mejorar la calidad del código de un programa.
- Informa sobre código duplicado, estándares de codificación, pruebas unitarias, cobertura de código, complejidad ciclomática, posible errores, comentarios y diseño del software.
- Aunque pensado para Java, acepta mas de 20 lenguajes mediante extensiones.
- Se integra con Maven, Ant y herramientas de integración continua como Atlassian Bamboo, Jenkins y Hudson).

Sonar



Análisis estático Web

- HTML
  - https://validator.w3.org/
- CSS
  - http://jigsaw.w3.org/css-validator/
- WAI
  - https://www.w3.org/WAI/ER/tools/
- JavaScript
  - http://jshint.com/
  - http://www.jslint.com/
- Chrome Dev Tools

Selenium Webdriver

INTRODUCCIÓN

Introducción

- El Selenium es un conjunto de herramientas para automatizar los navegadores web, robot que simula la interacción del usuario con el navegador, originalmente pensado como entorno de pruebas de software para aplicaciones basadas en la web.
- Como principales herramientas Selenium cuenta con:
  - Selenium IDE:
    - una herramienta para grabar y reproducir secuencias de acciones con el navegador que permite crear pruebas sin usar un lenguaje de scripting para pruebas.
  - Selenium Core:
    - API para escribir pruebas automatizadas y de regresión en un amplio número de lenguajes de programación populares incluyendo Java, C#, Ruby, Groovy, Perl, Php y Python.
  - WebDriver:
    - interfaces que permite ejecutar las pruebas de forma nativa usando la mayoría de los navegadores web modernos en diferentes sistemas operativos como Windows, Linux y OSX.
  - Selenium Grid:
    - Permite ejecutar muchas pruebas de un mismo grupo en paralelo o pruebas en múltiples entornos. Tiene la ventaja que un conjunto de pruebas muy grande puede dividirse en varias maquinas remotas para una ejecución más rápida o si se necesitan repetir las mismas pruebas en múltiples entornos.

Historia

- Cuando Selenium 1 fue lanzado en el año 2004, surgió por la necesidad de reducir el tiempo dedicado a verificar manualmente el comportamiento consistente en el front-end de una aplicación web. Hizo uso de las herramientas disponibles en ese momento y confió en gran medida en la inyección de JavaScript en la página web bajo prueba para emular la interacción de un usuario.
- Si bien JavaScript es una buena herramienta para permitirte la introspección de las propiedades del DOM y para hacer ciertas observaciones del lado del cliente que de otro modo no se podría hacer, se queda corto en la capacidad de replicar naturalmente las interacciones de un usuario como usar el mouse y el teclado.
- Desde entonces, Selenium ha crecido y ha madurado bastante, convirtiéndose en una herramienta ampliamente utilizada. Selenium ha pasado de ser de un kit de herramientas de automatización de pruebas de fabricación casera a la librería de facto de automatización de navegadores del mundo.
- Así como Selenium RC hizo uso de las herramientas de oficio disponibles en ese momento, Selenium WebDriver impulsa esta tradición al llevar la parte de la interacción del navegador al territorio del proveedor del mismo y pedirles que se responsabilicen de las implementaciones de back-end orientadas al navegador. Recientemente este esfuerzo se ha convertido en un proceso de estandarización del W3C donde el objetivo es convertir el componente WebDriver en Selenium en la librería de control remoto du jour para agentes de usuario.

Selenium controla los navegadores web

- Selenium significa muchas cosas pero en su núcleo, es un conjunto de herramientas para la automatización de navegadores web que utiliza las mejores técnicas disponibles para controlar remotamente las instancias de los navegadores y emular la interacción del usuario con el navegador.
- Permite al código simular interacciones básicas realizadas por los usuarios finales; insertando texto en los campos, seleccionando valores de menús desplegables y casillas de verificación, y haciendo clics en los enlaces de los documentos. También provee muchos otros controles tales como el movimiento del mouse, la ejecución arbitraria de JavaScript y mucho más.
- A pesar de que es usado principalmente para pruebas de front-end de sitios webs, Selenium es en esencia una librería de agente de usuario para el navegador. Las interfaces son ubicuas a su aplicación, lo que fomenta la composición con otras librerías para adaptarse a su propósito.

Un API para gobernarlos a todos

- Uno de los principios fundamentales del proyecto es permitir una interfaz común para todas las tecnologías de los (principales) navegadores.
- Los navegadores web son aplicaciones increíblemente complejas y de mucha ingeniería, realizando operaciones completamente diferentes pero que usualmente se ven iguales al hacerlo. Aunque el texto se presente con las mismas fuentes, las imágenes se muestren en el mismo lugar y los enlaces te llevan al mismo destino.
- Lo que sucede por debajo es tan diferente como la noche y el día. Selenium abstrae estas diferencias, ocultando sus detalles y complejidades a la persona que escribe el código. Esto le permite escribir varias líneas de código para realizar un flujo de trabajo complicado, pero estas mismas líneas se ejecutarán en Firefox, Internet Explorer, Chrome y los demás navegadores compatibles.

Herramientas y soporte

- El diseño minimalista de Selenium le da la versatilidad para que se pueda incluir como un componente en otras aplicaciones. La infraestructura proporcionada debajo del catálogo de Selenium te da las herramientas para que puedas ensamblar tu grid de navegadores de modo que tus pruebas se puedan ejecutar en diferentes navegadores a través de diferente sistemas operativos y plataformas.
- Imagina un granja de servidores en tu sala de servidores o en un centro de datos, todos ejecutando navegadores al mismo tiempo e interactuando con los enlaces en tu sitio web, formularios y tablas—probando tu aplicación 24 horas al día. A través de la sencilla interfaz de programación proporcionada para los lenguajes más comunes, estas pruebas se ejecutarán incansablemente en paralelo, reportando cuando ocurran errores.
- Es un objetivo ayudar a que esto sea una realidad para ti, proporcionando a los usuarios herramientas y documentación para controlar no solo los navegadores pero también para facilitar la escalabilidad e implementación de tales grids.

Automatización de pruebas

- Las pruebas funcionales de usuario final, como las pruebas de Selenium son caras de ejecutar, requieren abrir un navegador e interactuar con el. Además, normalmente requieren que una infraestructura considerable este disponible para estas ejecutarse de manera efectiva. Es una buena regla preguntarse siempre si lo que se quiere probar se puede hacer usando enfoques de prueba más livianos como las pruebas unitarias o con un enfoque de bajo nivel.
- Las pruebas manuales son muy costosas y difícilmente repetibles, por lo que se impone una estrategia de automatización. Selenium permite ejecutar y repetir las mismas pruebas en múltiples navegadores de diferentes sistemas operativos.
- Una vez tomada la decisión de hacer pruebas en el navegador, y que tengas tu ambiente de Selenium listo para empezar a escribir pruebas, generalmente realizaras alguna combinación de estos tres pasos:
  - Preparar los datos
  - Realizar un conjunto discreto de acciones
  - Evaluar los resultados
- Querrás mantener estos pasos tan cortos como sea posible; una o dos operaciones deberían ser suficientes la mayor parte del tiempo. La automatización del navegador tiene la reputación de ser "frágil", pero en realidad, esto se debe a que los usuarios suelen exigirle demasiado.
- Manteniendo las pruebas cortas y usando el navegador web solo cuando no tienes absolutamente ninguna alternativa, puedes tener muchas pruebas con fragilidad mínima.
- Una clara ventaja de las pruebas de Selenium es su capacidad inherente para probar todos los componentes de la aplicación, desde el backend hasta el frontend, desde la perspectiva del usuario. En otras palabras, aunque las pruebas funcionales pueden ser caras de ejecutar, también abarcan a la vez grandes porciones críticas para el negocio.

Tipos de pruebas

- **Pruebas funcionales**: Este tipo de prueba se realiza para determinar si una funcionalidad o sistema funciona correctamente y sin problemas. Se comprueba el sistema en diferentes niveles para garantizar que todos los escenarios están cubiertos y que el sistema hace lo que se supone que debe de hacer. Es una actividad de verificación que responde la pregunta: ¿Estamos construyendo el producto correctamente?.
  - Para aplicaciones web, la automatización de esta prueba puede ser hecha directamente con Selenium simulando los retornos esperados. Esta simulación podría hacerse mediante grabación/reproducción o mediante los diferentes lenguajes soportados.
- **Pruebas de aceptación**: Este tipo de prueba se realiza para determinar si una funcionalidad o un sistema cumple con las expectativas y requerimientos del cliente. Este tipo de pruebas implican la cooperación o retroalimentación del cliente, siendo una actividad de validación que responde la pregunta: ¿Me están construyendo el producto correcto?.
  - Las pruebas de aceptación son un subtipo de pruebas funcionales, por lo que la automatización se puede hacer directamente con Selenium simulando el comportamiento esperado del usuario mediante grabación/reproducción.

Tipos de pruebas

- **Pruebas de regresión**: Este tipo de pruebas generalmente se realiza después de un cambio, corrección o adición de funcionalidad.
  - Para garantizar que el cambio no ha roto ninguna de las funcionalidades existentes, algunas pruebas ya ejecutadas se ejecutan nuevamente. El conjunto de pruebas ejecutadas nuevamente puede ser total o parcial, y puede incluir varios tipos diferentes, dependiendo del equipo de desarrollo y la aplicación.
- Las **pruebas no funcionales** tales como rendimiento, de carga, de estrés, … aunque se pueden realizar con Selenium hay otras opciones, como Jmeter, que las realizan mas eficientemente y además suministran métricas que incluyen rendimiento, latencia, pérdida de datos, tiempos de carga de componentes individuales.
- De igual forma, las **pruebas unitarias** se pueden realizar con Selenium pero, como se deben ejecutar continuamente, suele ser demasiado costoso.

Proceso de Prueba

- Descomponer la aplicación web existente para identificar qué probar.
- Identificar con qué navegadores probar.
- Elige el mejor lenguaje para ti y tu equipo.
- Configura Selenium para que funcione con cada navegador que te interese.
- Escriba pruebas de Selenium mantenibles y reutilizables que serán compatibles y ejecutables en todos los navegadores.
- Crea un circuito de retroalimentación integrado para automatizar las ejecuciones de prueba y encontrar problemas rápidamente.
- Configura tu propia infraestructura o conéctate a un proveedor en la nube.
- Mejora drásticamente los tiempos de prueba con la paralelización.
- Mantente actualizado en el mundo Selenium.

Que probar

- La navegación
- La interacciones con la página y sus elementos
- El rellenado de formularios y sus validaciones
- El arrastrar y soltar, si procede
- La estética, presencia y visualización de elementos esenciales, con especial atención al responsive design.
- Moverse entre ventanas y marcos (aunque están prohibidos por la WAI)
- Uso de cookies, Local Storage, Session Storage, Service Worker, …

INSTALACIÓN

Drivers

- Instalar los WebDriver:
  - Crear una carpeta y referenciarla en el PATH del sistema.
    - Windows: *setx /m path "%path%;C:\Selenium\WebDriver\"*
    - macOS y Linux: *export PATH=$PATH:/opt/Selenium/WebDriver >> ~/.profile*
  - Descargar los drivers (https://www.selenium.dev/downloads/)
  - Copiarlos descomprimidos en la carpeta creada.
- Plataformas compatibles con Selenium:
  - Firefox: GeckoDriver está implementado y soportado por Mozilla.
  - Internet Explorer: Solo se admite la versión 11 y requiere una configuración adicional.
  - Safari: SafariDriver es compatible directamente con Apple.
  - Ópera: OperaDriver es compatible con Opera Software.
  - Chrome: ChromeDriver es compatible con el proyecto Chromium.
  - Edge: EdgeDriver está implementado y soportado por Microsoft.

Local

- Instalación de Eclipse.
- Descargar y descomprimir API: https://www.selenium.dev/downloads/
- Creación del proyecto:
  - Create a Java Project
    - Add Library Junit 5
    - Add External JARs: API descomprimida
  - Crear paquete de tests
    - Añadir JUnit Test Case
  - Ejecutar test
    - Run As → JUnit Test Case.

Maven

- Instalación de Eclipse.
- Creación del proyecto:
  - Create a New Maven Project
    - Create a simple project (skip archetype selection)
  - Editar pom.xml
    - Añadir dependencias
  - Seleccionar src/test/java
    - Añadir JUnit Test Case
  - Ejecutar test
    - Run As → JUnit Test Case.

pom.xml

```
<dependency>
	<groupId>org.junit.jupiter</groupId>
	<artifactId>junit-jupiter</artifactId>
	<version>${junit.jupiter.version}</version>
	<scope>test</scope>
</dependency>
<dependency>
	<groupId>org.seleniumhq.selenium</groupId>
	<artifactId>selenium-java</artifactId>
	<version>${selenium.version}</version>
	<scope>test</scope>
</dependency>
```

Dependencias únicas

- La dependencia selenium-java permite la ejecución del proyecto de automatización en todos los navegadores compatibles con Selenium.
- Si solo es necesario ejecutar las pruebas en un navegador en específico, se puede sustituir dependencia selenium-java por la dependencia para ese navegador en el archivo pom.xml. Por ejemplo, se debe agregar la siguiente dependencia en el archivo pom.xml para ejecutar las pruebas solamente en Chrome:

```
<dependency>
  <groupId>org.seleniumhq.selenium</groupId>
  <artifactId>selenium-chrome-driver</artifactId>
  <version>3.X</version>
</dependency>
```


SELENIUM IDE

Introducción

- Es el entorno de desarrollo integrado para pruebas con Selenium que permite grabar, editar y depurar fácilmente las pruebas. Es una solución simple y llave en mano para crear rápidamente pruebas de extremo a extremo confiables.
- Selenium IDE no requiere una configuración adicional aparte de instalar una extensión en el navegador pero solo está disponible para Firefox y Chrome.
- Selenium IDE es muy flexible, registra múltiples localizadores para cada elemento con el que interactúa, si un localizador falla durante la reproducción, los demás se probarán hasta que uno tenga éxito.
- Mediante el uso del comando de ejecución se puede reutilizar un caso de prueba dentro de otro.
- Dispone una estructura de flujo de control extensa, con comandos condicionales, bucles, … que permite modelizar escenarios complejos.

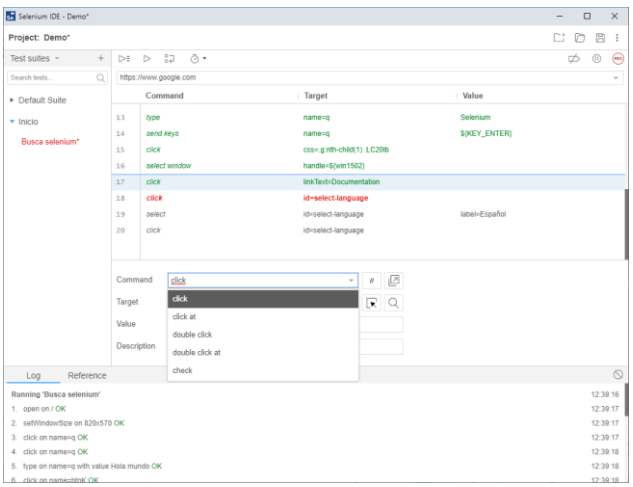
Características

- Dispone de una selección inteligente de campos usando ID, nombre, Xpath o DOM según se necesite.
- Para la depuración permite la configuración de los puntos de interrupción, iniciar y detener la ejecución de un caso de prueba desde cualquier punto dentro del caso de prueba e inspeccionar la forma en el caso de prueba se comporta en ese punto.
- Permite exportar los casos de prueba a Java, C# y Ruby, actuando como embriones en la creación de los casos de prueba para WebDriver.
- Selenium IDE dispone de un amplio conjunto de extensiones adicionales que ayudan o simplifican la elaboración de los casos de pruebas.

Empezando

- Instale Selenium IDE desde la tienda web Chrome o Firefox .
- Una vez instalado, inícielo haciendo clic en su icono desde la barra de menú de su navegador.
- Aparecerá un cuadro de diálogo de bienvenida con las siguientes opciones:
  - Grabar una nueva prueba en un nuevo proyecto
  - Abrir un proyecto existente
  - Crea un nuevo proyecto
  - Cierra el IDE
- Si se da grabar, pedirá el nombre del proyecto y la URL base de la aplicación que se está probando (se utiliza en todas las pruebas del proyecto).
- Después de completar esta configuración, se abrirá la ventana IDE y una nueva ventana del navegador, cargará la URL base y comenzará a grabar. Interactúe con la página y cada una de las acciones se registrará en el IDE. Para detener la grabación, cambie a la ventana IDE y haga clic en el icono de grabación.

Empezando



Empezando

- La grabación se puede editar, modificar, reproducir, depurar, guardar y exportar.
- Cada grabación es una prueba. El proyecto puede contener múltiples pruebas que se organizan agrupándolas en suites. Por defecto se crea la “Default Suite” a la que se le asigna la prueba de la grabación inicial. Las pruebas y las suites se pueden renombrar, ejecutar o eliminar.
- El proyecto se puede descartar o guardar un único archivo JSON con una extensión .side para recuperarlo posteriormente o ejecutarlo en la línea de comandos.
- La reproducción de la prueba en el IDE permite establecer puntos de ruptura e inspeccionar valores.
- Se puede reanudar la grabación en cualquier punto.

Scripts

- Se pueden desarrollar automáticamente scripts al crear una grabación y de esa manera se puede editar manualmente con sentencias y comandos para que la reproducción de nuestra grabación sea correcta
- Los scripts se generan en un lenguaje de scripting especial para Selenium a menudo denominado Selanese.
- Selanese provee comandos que dicen al Selenium que hacer y pueden ser:
  - **Acciones**: son comandos que generalmente manipulan el estado de la aplicación, ejecutan acciones sobre objetos del navegador, como hacer click en un enlace, escribir en cajas de texto o seleccionar de una lista de opciones. Muchas acciones pueden ser llamadas con el sufijo "AndWait“ que indica la acción hará que el navegador realice una llamada al servidor y que se debe esperar a una nueva página se cargue.
  - **Descriptores de acceso**: examinan el estado de la página y almacenan los resultados en variables.
  - **Aserciones**: son como descriptores de acceso, pero las muestras confirman que el estado de la solicitud se ajusta a lo que se esperaba, verifican la presencia de un texto en particular o la existencia de elementos.

Localizadores

- Localizar por Id:
  - id=loginForm
- Localizar por Name
  - name=username
- Localizar por XPath
  - xpath=//form[@id='loginForm']
- Localizar por el texto en los hipervínculos
  - link=Continue
- Localizar por CSS
  - css=input[name="username"]

Acciones

- Sobre elementos de los formularios:
  - click, click at, double click, double click at, check, uncheck, edit content, send keys, type, select, add selection, remove selection, submit
- Del ratón:
  - mouse move at, mouse over, mouse out, mouse down, mouse down at, mouse up, mouse up at, drag and drop to object
- Sobre cuadros de dialogo alert, prompt y confirm:
  - choose ok on next confirmation, choose cancel on next confirmation, choose cancel on next prompt, webdriver choose ok on visible confirmation, webdriver choose cancel on visible confirmation, webdriver choose cancel on visible prompt
- Sobre la ejecución de la prueba:
  - open, execute script, run, echo, debugger, pause, close

Variables

- Se puede usar variable en Selenium para almacenar constantes al principio de un script. Además, cuando se combina con un diseño de prueba controlado por datos, las variables de Selenium se pueden usar para almacenar valores pasados a la prueba desde la línea de comandos, desde otro programa o desde un archivo.
  - store target:valor value:varName
- Para acceder al valor de una variable:
  - ${userName}
- Hay métodos diponibles para recuperar información de la página y almacenarla en variables:
  - storeAttribute, storeText, storeValue, storeTitle, storeXpathCount

Flujo de control

- Selenium IDE viene con comandos que permiten agregar lógica condicional y bucles a las pruebas, para ejecutar comandos (o un conjunto de comandos) solo cuando se cumplen ciertas condiciones o repetidamente en función de criterios predefinidos.
- Las condiciones en su aplicación se verifican mediante el uso de expresiones de JavaScript.
- Los comandos de flujo de control Control Flow funcionan son comandos de apertura y cierre para denotar un conjunto (o bloque) de comandos.
- Aquí están cada uno de los comandos de flujo de control disponibles acompañados de sus comandos complementarios de cierre.
  - if … else if … else … end // else if y else son opcionales
  - times … end // Bucle for
  - forEach … end
  - while … end
  - do … repeat if // la condicion en repeat if
- Se pueden anidar los comandos de control de flujo según sea necesario.

Afirmar y Verificar

- Una "afirmación" hará fallar la prueba y abortará el caso de prueba actual, mientras que una "verificación" hará fallar la prueba pero continuará ejecutando el caso de prueba.
  - Tiene muy poco sentido para comprobar que el primer párrafo de la página sea el correcto si la prueba ya falló al comprobar que el navegador muestra la página esperada. Por otro lado, es posible que desee comprobar muchos atributos de una página sin abortar el caso de prueba al primer fallo, ya que esto permitirá revisar todas los fallos en la página y tomar la acción apropiada.
- Selenese permite múltiples formas de comprobar los elementos de la interfaz de usuario pero hay que decidir el métodos mas apropiado:
  - ¿Un elemento está presente en algún lugar de la página?
  - ¿El texto especificado está en algún lugar de la página?
  - ¿El texto especificado está en una ubicación específica en la página?
- Métodos:
  - assert, assertAlert, assertChecked, assertNotChecked, assertConfirmation, assertEditable, assertNotEditable, assertElementPresent, assertElementNotPresent, assertPrompt, assertSelectedValue, assertNotSelectedValue, assertSelectedLabel, assertText, assertNotText, assertTitle, assertValue
  - verify, verifyChecked, verifyNotChecked, verifyEditable, verifyNotEditable, verifyElementPresent, verifyElementNotPresent, verifySelectedValue, verifyNotSelectedValue, verifyText, verifyNotText, verifyTitle, verifyValue, verifySelectedLabel

Ejecutar en línea de comandos

- Requiere tener instalado NodeJS (https://nodejs.org)
- Instalar CLI
  - npm install -g selenium-side-runner
- Instalar los WebDriver:
  - Crear una carpeta y referenciarla en el PATH del sistema.
  - Descargar los drivers (https://www.selenium.dev/downloads/)
  - Copiarlos a la carpeta creada.
- Para ejecutar las suites de pruebas:
  - selenium-side-runner project.side project2.side *.side
- Para ejecutar en diferentes navegadores:
  - selenium-side-runner *.side -c "browserName=Chrome"
  - selenium-side-runner *.side -c "browserName=firefox"

Exportación

- Se puede exportar una prueba o un conjunto de pruebas a código de WebDriver haciendo clic con el botón derecho en una prueba o un conjunto, seleccionando Export, eligiendo el idioma de destino y haciendo clic Export.
- Actualmente, se admite la exportación a los siguientes idiomas y marcos de prueba.
  - C# NUnit
  - C# xUnit
  - Java JUnit
  - JavaScript Mocha
  - Python pytest
  - Ruby RSpec
- Esta previsto admitir todos los lenguaje de programación soportados oficialmente para Selenium en al menos un marco de prueba para cada lenguaje.

WebDriver

```
@BeforeClass
public static void setUpClass() throws Exception {
        System.setProperty("webdriver.chrome.driver", "C:/Archivos/.../chromedriver.exe");
}
@Before
public void setUp() throws Exception {
  driver = new ChromeDriver();
  baseUrl = "http://localhost/";
  driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
}
@Test
public void testLoginOK() throws Exception {
  driver.get(baseUrl + "/login.php");
  driver.findElement(By.id("login")).sendKeys("admin");
  driver.findElement(By.id("password")).sendKeys("admin");
  driver.findElement(By.cssSelector("input[type=\"submit\"]")).click();
  try {
    assertEquals("", driver.findElement(By.cssSelector("img[title=\"Main Menu\"]")).getText());
  } catch (Error e) {
    verificationErrors.append(e.toString());
  }
}
```

Maven

```
<dependency>
  <groupId>org.seleniumhq.selenium</groupId>
  <artifactId>selenium-java</artifactId>
  <version>3.13.0</version>
</dependency>
```

WEBDRIVER

WebDriver

- Selenium permite la automatización de todos los principales navegadores del mercado mediante el uso de WebDriver.
- WebDriver es una API y un protocolo que define una interfaz de idioma neutral para controlar el comportamiento de los navegadores web. Cada navegador está respaldado por un Driver, una implementación específica de WebDriver, llamada controlador. El controlador es el componente responsable de delegar en el navegador, y maneja la comunicación entre Selenium y el navegador.
- Esta separación es parte de un esfuerzo consciente para hacer que los proveedores de navegadores asuman la responsabilidad de la implementación para sus navegadores. Selenium utiliza estos controladores de terceros cuando es posible, pero también proporciona sus propios controladores mantenidos por el proyecto para los casos en que esto no es una realidad.
- El framework de Selenium unifica todas estas piezas a través de una interfaz orientada al usuario que habilita que los diferentes backends de los navegadores sean utilizados de forma transparente, permitiendo la automatización cruzada entre navegadores y plataformas diferentes.

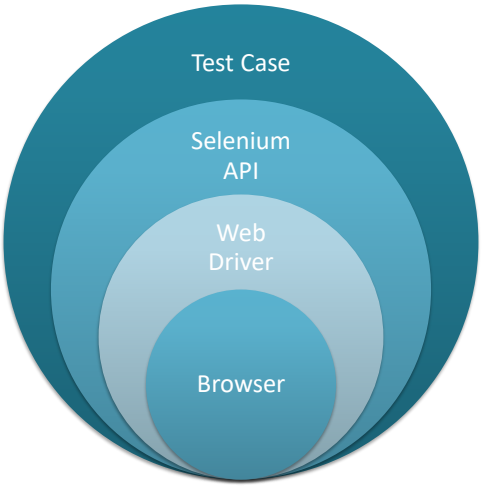
WebDriver

- WebDriver controla un navegador de forma nativa, como lo haría un usuario, ya sea localmente o en una máquina remota utilizando el servidor Selenium, marca un salto adelante en términos de automatización de navegadores.
- Selenium WebDriver se refiere tanto a los enlaces de lenguajes como también a las implementaciones individuales del código controlador del navegador. Esto se conoce comúnmente solo como WebDriver.
- Selenium WebDriver es una Recomendación W3C (https://www.w3.org/TR/webdriver1/)
  - WebDriver está diseñado como una interfaz de programación simple y más concisa.
  - WebDriver es una API compacta orientada a objetos.
  - Controla el navegador de manera efectiva.

Componentes

- API: Interfaz de Programación de Aplicaciones. Es un conjunto de “comandos” que se utilizan para manipular el WebDriver.
- Library: Un módulo de código que contiene las APIs y el código necesario para implementarlos. Las librerías son específicas para cada lenguaje, por ejemplo ficheros .jar en Java, ficheros .dll para .NET, etc.
- Driver: El responsable de controlar un navegador concreto (proxies). La mayoría de los drivers son creados por los fabricantes del navegador. Los Drivers son generalmente módulos ejecutables que corren en el sistema con el propio navegador, no en el sistema ejecutando la suite de test.
- Remote WebDrive: Permite controlar instancias remotas de navegadores web.
- Framework: Una librería adicional usada como un soporte para la suites de WebDriver. Estos frameworks pueden ser test frameworks como JUnit o NUnit. También pueden ser frameworks soportando lenguaje natural como Cucumber o Robotium. Los frameworks también pueden ser escritos y usados para cosas como la manipulación o configuración del sistema bajo la prueba, creación de datos, test oracles, etc

Componentes



Navegadores

El framework de Selenium soporta oficialmente los siguientes navegadores:

Navegador	Proveedor	Versiones Soportadas
Chrome	Chromium	Todas las Versiones
Firefox	Mozilla	54 y más recientes
Internet Explorer	Selenium	6 y más recientes
Opera	Opera Chromium / Presto	10.5 y más recientes
Safari	Apple	10 y más recientes

También hay un conjunto de navegadores especializados utilizados típicamente en entornos de desarrollo.

Controlador	Propósito	Mantenedor
HtmlUnitDriver	Emulador de navegador headless respaldado por Rhino	Proyecto Selenium

Navegadores simulados

- HtmlUnit
  - HtmlUnit es un navegador sin interfaz grafica para programas basados en Java. Modela documentos HTML y proporciona un API que permite invocar las paginas, rellanar formularios, hacer clic en enlaces, etc. Soporta JavaScript y es capaz de funcionar con librerías AJAX, simulando Chrome, Firefox o Internet Explorer dependiendo de la configuración usada.
- PhantomJS
  - PhantomJS es un navegador sin interfaz grafica basado en Webkit, aunque es una versión mucho mas antigua que las usadas por Chrome o Safari. El proyecto esta sin soporte desde que se anunció que Chrome y Firefox tendrían la capacidad de ser navegadores sin interfaz grafica.

Instalación

- La gran mayoría de controladores necesitan de un ejecutable extra para que Selenium pueda comunicarse con el navegador. Para ejecutar tu proyecto y controlar el navegador, debes tener instalados los binarios de WebDriver específicos para el navegador.
- Crea un directorio para almacenar los ejecutables en el, como C:\WebDriver\bin o /opt/WebDriver/bin. A través de los diferentes proveedores, descargar y descomprimir en la carpeta recién creada.
- Se puede especificar manualmente donde esta ubicado el ejecutable antes lanzar el WebDriver, pero esto hará que los tests sean menos portables, ya que los ejecutables necesitan estar en el mismo lugar en todas las maquinas. Si la ruta de la carpeta está incluida en el PATH no es necesario especificarla en cada test:
  - Windows: setx /m path "%path%;C:\WebDriver\bin\"
  - macOS y Linux: export PATH=$PATH:/opt/WebDriver/bin >> ~/.profile
- Para probar los cambios, en una terminal de comando escribe el nombre de uno de los binarios que has añadido en la carpeta en el paso previo:
  - chromedriver

Cargar el driver

- Importar el espacio de nombres de WebDriver.
  ```
  import org.openqa.selenium.WebDriver;
  ```
- Si la ruta a los dirvers está en el PATH no debe fijarse con System.setProperty.
- Chromium/Chrome
  ```
  import org.openqa.selenium.chrome.ChromeDriver;
  System.setProperty("webdriver.chrome.driver", "/path/to/chromedriver");
  WebDriver driver = new ChromeDriver();
  ```
- Firefox
  ```
  import org.openqa.selenium.Firefox.FirefoxDriver;
  System.setProperty("webdriver.gecko.driver", "/path/to/geckodriver");
  WebDriver driver = new FirefoxDriver();
  ```
- Edge
  ```
  import org.openqa.selenium.edge.EdgeDriver;
  System.setProperty("webdriver.edge.driver", "C:/path/to/MicrosoftWebDriver.exe");
  WebDriver driver = new EdgeDriver();
  ```
- Internet Explorer
  ```
  import org.openqa.selenium.ie.InternetExplorerDriver;
  System.setProperty("webdriver.ie.driver", "C:/path/to/IEDriver.exe");
  WebDriver driver = new InternetExplorerDriver();
  ```


Estrategia de carga de página

- pageLoadStrategy permite los siguientes valores:
  - NORMAL (por defecto) : Esto hará que WebDriver espere a que se cargue toda la página. Cuando se establece en NORMAL, WebDriver espera hasta que se dispare el evento load y concluya.
  - EAGER: Esto hará que WebDriver espere hasta que el documento HTML inicial se haya cargado y analizado por completo, descartando la carga de hojas de estilo, imágenes y sub marcos. Cuando se establece en EAGER, WebDriver espera hasta que se dispare el evento DOMContentLoaded y concluya.
  - NONE: Esto hará que WebDriver solo espera hasta que se descargue la página inicial.

```
ChromeOptions chromeOptions = new ChromeOptions();
chromeOptions.setPageLoadStrategy(PageLoadStrategy.EAGER);
WebDriver driver = new ChromeDriver(chromeOptions);
```

Aislar las pruebas

- Hay muchos recursos externos que se cargan en una página que no son directamente relevantes para la funcionalidad que está probando (por ejemplo, widgets de Facebook, Analytics, fragmentos de JavaScript, CDN, etc.). Estos recursos externos tienen el potencial de impactar negativamente las ejecuciones de prueba debido a tiempos de carga lentos.
- Entonces, ¿cómo proteger nuestras pruebas de estas cosas que están fuera de su control?
- Una posible solución es utilizar un servidor proxy en nuestras pruebas, que actúe como un doble de pruebas, donde podremos bloquear los recursos externos que no queremos cargar agregándolos a una lista negra o suministrando una versión ligera.

Proxy

- Un servidor proxy actúa como intermediario para solicitudes entre un cliente y un servidor. En forma simple, el tráfico fluye a través del servidor proxy en camino a la dirección solicitada y de regreso.
- Un servidor proxy para scripts de automatización con Selenium podría ser útil para:
  - Capturar el tráfico de la red
  - Simular llamadas de backend realizadas por el sitio web
  - Accede al sitio web requerido bajo topologías de red complejas o restricciones/políticas corporativas estrictas.
- Si te encuentras en un entorno corporativo, y un navegador no puede conectarse a una URL, esto es muy probablemente porque el ambiente necesita un proxy para acceder.
- Selenium WebDriver proporciona una vía para configurar el proxy:

```
Proxy proxy = new Proxy();
proxy.setHttpProxy("<HOST:PORT>");
ChromeOptions options = new ChromeOptions();
options.setCapability("proxy", proxy);
WebDriver driver = new ChromeDriver(options);
```

Perfiles Firefox

- Firefox guarda tu información personal (marcadores, contraseñas y preferencias de usuario) en un conjunto de archivos llamado perfil, que se almacena en una ubicación de disco diferente a la que se utiliza para almacenar los archivos de ejecución de Firefox. Puedes crear distintos perfiles, cada uno de ellos con un conjunto de datos de usuario diferente. Solo puede estar activo un perfil en un momento determinado. Mediante el Administrador de perfiles de Firefox puedes crear, eliminar y renombrar los perfiles.
- Cuando se desee ejecutar una automatización confiable en un navegador Firefox, se recomienda crear un perfil separado.
- Para iniciar el Administrador de perfiles se introduce en la barra de about:profiles.
- Para abrir Firefox con un determinado perfil: firefox.exe -P webdrive

```
ProfilesIni profile = new ProfilesIni();
FirefoxProfile myprofile = profile.getProfile("testProfile");
WebDriver driver = new FirefoxDriver(myprofile);
```


Visión de conjunto

- Una vez referenciado el controlador ya se puede interactuar con el navegador, todas las acciones se realizaran a través de dicha referencia.
- Con el controlador se navega a una nueva página.
- En la página, se buscaran los diferentes elementos (tag) con el método findElement y un localizador, que devolverá un WebElement. Este método recupera un solo elemento, si se necesita recuperar varios elementos, el método findElements obtiene la colección de elementos localizados.
- Un localizador es un objeto que define el selector de los elementos web basándose en diferentes estrategia como ID, Nombre, Clase, XPath, Selectores CSS, Texto de enlace, etc. Los WebElements se pueden encontrar buscando desde la raíz del documento o buscando en otra WebElement.
- Un WebElement representa un elemento del DOM. El WebElement tiene un conjunto de métodos de acción, tales como click(), getText() y sendKeys().
- Esta es la forma principal para interactuar con un elemento (etiqueta) de la página y obtener información de respuesta de el.

Navegación

- Navegar hacia
  driver.get("https://selenium.dev"); // Recomendada
  driver.navigate().to("https://selenium.dev");
- Retroceder
  driver.navigate().back();
- Avanzar
  driver.navigate().forward();
- Actualizar
  driver.navigate().refresh();
- Obtener la URL actual
  url = driver.getCurrentUrl();
- Obtener el título
  driver.getTitle();

Navegación

- Salir y cerrar el navegador al final de una sesión
  driver.quit();
  - Muy importante porque:
    - Cerrará todas las ventanas y pestañas asociadas a esa sesión del WebDriver.
    - Cerrará el proceso de navegador.
    - Cerrará el proceso en segundo plano del driver.
    - Notificará al Grid de Selenium que el navegador ya no esta en uso y que puede ser usado por otra sesión del Grid de Selenium.
  - Un fallo en la llamada del método salir dejará procesos corriendo en segundo plano y puertos abiertos en tu maquina lo que podría llevar a problemas en un futuro.

Estrategia de carga de página

- Por defecto, cuando WebDriver carga una página, sigue la estrategia de carga NORMAL. Siempre se recomienda detener la descarga de más recursos adicionales (como imágenes, css, js) cuando la carga de la página lleva mucho tiempo.
- La propiedad document.readyState de un documento describe el estado de carga del documento actual. Por defecto, WebDriver esperará responder a una llamada driver.get() o driver.navigate().to() hasta que el estado de documento listo esté completo
- En aplicaciones SPA (como Angular, react, Ember) una vez que el contenido dinámico ya está cargado (es decir, una vez que el estado de readyState es COMPLETE), hacer clic en un enlace o realizar alguna acción dentro de la página no disparara una nueva solicitud al servidor ya que el contenido se carga dinámicamente en el lado del cliente sin una actualización de la página. Las aplicaciones de SPA pueden cargar muchas vistas dinámicamente sin ninguna solicitud al servidor, por lo que pageLoadStrategy siempre mostrará el estado 'COMPLETE' hasta que se haga un nuevo driver.get() y driver.navigate().to().

Cookies

- Una cookie es una pequeña pieza de datos que es enviada desde el sitio web y es almacenada en el ordenador. Las cookies son usadas principalmente para reconocer al usuario y cargar la información personalizada.
- Se gestionan a traves de driver.manage(), una ver descargada la pagina:

```
driver.manage().addCookie(new Cookie("key", "value"));
Cookie cookie = driver.manage().getCookieNamed("foo");
Set<Cookie> cookies = driver.manage().getCookies();
driver.manage().deleteCookieNamed("foo");
driver.manage().deleteCookie(cookie);
driver.manage().deleteAllCookies();
```

- Por defecto, cuando el atributo sameSite esta fijado como Strict (estricto en español), la cookie no será enviada junto a las peticiones iniciadas por paginas web externas. Cuando se fija como Lax, será enviada junto con la petición GET iniciada por paginas web externas.

```
Cookie cookie = new Cookie.Builder("key", "value").sameSite("Strict").build();
Cookie cookie1 = new Cookie.Builder("key", "value").sameSite("Lax").build();
```

Ventanas, solapas, Frames e Iframes

- El uso de múltiples ventanas, o solapas, Frames e Iframes está prohibido por la reglas de WAI, dado construyem sitios desde múltiples documentos en el mismo dominio.
- Para trabajar con ventanas o solapas:

```
driver.getWindowHandle();
driver.switchTo().window(windowHandle);
driver.close();
```

- Para trabajar con Frames e Iframes

```
driver.switchTo().frame("myframe");
driver.switchTo().defaultContent();
```


Tamaño del navegador

- La resolución de las pantallas puede impactar en como la aplicación se renderiza.
- El W3C impone restricciones sobre los elementos presentes pero no visibles.
- WebDriver provee de mecanismos para mover y cambiar el tamaño de la ventana del navegador:

```
driver.manage().window().setSize(new Dimension(1024, 768));
driver.manage().window().maximize();
driver.manage().window().minimize();
driver.manage().window().fullscreen();
```

Buscar elementos

- Los WebElements se pueden encontrar buscando desde la raíz del documento utilizando una instancia de WebDriver o buscando en otra WebElement. Las búsquedas son costosas por lo que conviene guardar el resultado para no repetirlas.
- Localizar un elemento o el primer elemento:

```
WebElement searchForm = driver.findElement(By.id("myForm"));
WebElement searchBox = searchForm.findElement(By.name("q"));
```

- Localizar múltiples elementos:

```
List<WebElement> elements = driver.findElements(By.tagName("p"));
```

- Localizar el elemento activo (que tiene el foco en el contexto de navegación actual).

```
WebElement active = driver.SwitchTo().ActiveElement();
```


Localización de elementos

- La interfaz By encapsula las diferentes estrategias de localización de elementos. Hay ocho estrategias diferentes de ubicación de elementos integradas en WebDriver.

- By.id: Localiza elementos cuyo atributo ID coincide con el valor de la búsqueda

```
WebElement tag = driver.findElement(By.id("myForm"));
```

- By.name: Localiza elementos cuyo atributo NAME coincide con el valor de la búsqueda

```
WebElement tag = driver.findElement(By.name("username"));
```

- By.className: Localiza elementos en el que el nombre de su clase contiene el valor de la búsqueda (no se permiten nombres de clase compuestos)

```
WebElement tag = driver.findElement(By.className("container-fluid"));
```

Localización de elementos

- By.cssSelector: Localiza elementos que coinciden con un selector CSS

  WebElement tag = driver.findElement(By.cssSelector(".header img"));
- By.linkText: Localiza hipervínculos cuyo texto visible coincide con el valor de búsqueda

  WebElement tag = driver.findElement(By.linkText("Close"));
- By.partialLinkText: Localiza hipervínculos cuyo texto visible coincide con el valor de búsqueda

  WebElement tag = driver.findElement(By.partialLinkText("Next"));
- By.tagName: Localiza elementos cuyo nombre de etiqueta (tagName) coincide con el valor de búsqueda

  List<WebElement> elements = driver.findElements(By.tagName("img"));
- By.xpath : Localiza elementos utilizando una expresión Xpath

  WebElement tag = driver.findElement(By.xpath("//div[2]/input"));

Localización de elementos

- En general, si los ID o Name del HTML están disponibles, son únicos y predecibles, son el método preferido para ubicar un elemento en una página. Tienden a trabajar muy rápido y evitan costosos recorridos DOM.
- Si las ID únicas no están disponibles, un selector CSS bien escrito es el método preferido para localizar un elemento. XPath funciona tan bien como los selectores CSS, pero la sintaxis es complicada y con frecuencia difícil de depurar. Aunque los selectores XPath son muy flexibles, generalmente su desempeño no es probado por lo proveedores de navegadores y tienden a ser bastante lentos.
- Las estrategias de selección basadas en enlaces de texto y enlaces de texto parciales tienen el inconveniente en que solo funcionan en elementos de enlace. Además, internamente llaman a los selectores XPath.
- El nombre de la etiqueta puede ser una forma peligrosa de localizar elementos. Existen frecuentemente múltiples elementos con la misma etiqueta presentes en la página. Es útil sobre todo cuando se llama al método findElements(By) que devuelve una colección de elementos.
- La recomendación es mantener los localizadores tan compactos y legibles como sea posible. Pedirle a WebDriver que atraviese la estructura del DOM es una operación costosa y, cuanto más se pueda reducir el alcance de tu búsqueda, mejor.

Operar con elementos

- Recuperar el estado del elemento

  .getText(): Obtiene el texto visible (es decir, no oculto por CSS) del elemento, incluidos los subelementos.

  .isDisplayed(): Determina si el elemento es visible. Este método evita el problema de tener que analizar el atributo "estilo" de un elemento.

  .isEnabled(): Determina si el elemento está habilitado actualmente.

  .isSelected(): Determina si este elemento está seleccionado, tiene el foco.

  .getAttribute(): Obtiene el valor del atributo dado del elemento.

  .getCssValue(): Obtiene el valor de una propiedad CSS dada.

  .getLocation(): Obtiene en qué parte de la página se encuentra la esquina superior izquierda del elemento renderizado.

  .getSize(): Obtiene el ancho y el alto del elemento renderizado

  .getTagName(): Obtiene el nombre de la etiqueta de este elemento.

```
assertEquals("WebDriver", driver.findElement(By.id("titulo")).getText());
```


Operar con elementos

- Acciones
  - .click(): Haz clic en este elemento.
  - .sendKeys(): Simula escribir en un elemento.
  - .clear(): Si es un elemento de entrada de texto, borra el valor.
  - .submit(): Si el elemento es un formulario o un elemento dentro de un formulario, se enviará el formulario al servidor remoto.

```
driver.findElement(By.name("password")).clear();
driver.findElement(By.name("password")).sendKeys("god");
driver.findElement(By.id("myForm")).submit();
```

- Buscar subelementos:
  - .findElement(): Encuentra el primer WebElement usando el localizador dado.
  - .findElements(): Encuentra todos los elementos dentro del elemento usando el localizador dado.

Esperas

- Generalmente se puede decir que WebDriver posee una API de bloqueo. Porque es una biblioteca fuera-de-proceso que instruye al navegador qué hacer, y debido a que la plataforma web tiene una naturaleza intrínsecamente asíncrona, WebDriver no rastrea el estado activo y en tiempo real del DOM.
- Afortunadamente, el conjunto normal de acciones disponibles en la interfaz WebElement tales como click o sendKeys estan garantizadas para ser síncronas, es decir, las llamadas a métodos no vuelven hasta que el comando se haya completado en el navegador. Las API avanzadas de interacción del usuario, Keyboard y Mouse, son excepciones ya que están explícitamente pensadas como comandos asíncronos “Haz lo que te digo”.
- Esperar es hacer que transcurra una cierta cantidad de tiempo antes de continuar con el siguiente paso en la ejecución automatizada de la tarea.

Esperas explícitas

- Permiten que el código detenga la ejecución del programa, o congelar el hilo, hasta que la condición pasada sea resuelva. La condición se llama con cierta frecuencia, hasta que transcurra el tiempo de espera. Esto significa que mientras la condición devuelva un valor falso, seguirá intentando y esperando.
- Dado que las esperas explícitas permiten esperar a que ocurra una condición, hacen una buena combinación para sincronizar el estado entre el navegador, y su DOM, con el código de WebDriver.
- Para esperar a que la llamada findElement espere hasta que el elemento agregado dinámicamente desde el script se haya agregado al DOM:

```
WebElement firstResult = new WebDriverWait(driver, 10)
      .until(driver -> driver.findElement(By.xpath("//li/a")));
assertEquals(firstResult.getText());
```

- La condición de espera se puede personalizar para optimizar la espera:

```
WebElement firstResult = new WebDriverWait(driver, 10)
      .until(ExpectedConditions.elementToBeClickable(By.xpath("//li/a")));
```

Condiciones esperadas

- Es una necesidad bastante común tener que sincronizar el DOM con el código de prueba, hay condiciones predefinidas para operaciones frecuentes de espera.
- Las condiciones disponibles varían en las diferentes librerías de los lenguajes, pero esta es una lista no exhaustiva de algunos:
  - alert is present (la alerta esta presente)
  - element exists (el elemento existe)
  - element is visible (el elemento es visible)
  - title contains (el titulo contiene)
  - title is (el titulo es)
  - visible text (el texto es visible)
  - text to be (el texto es)

Espera predefinida

- Una instancia de FluentWait define la cantidad máxima de tiempo para esperar por una condición, así como la frecuencia con la que verificar dicha condición.
- Se puede configurar la espera para ignorar tipos específicos de excepciones mientras esperan, como NoSuchElementException cuando buscan un elemento en la página.
- La instancia de FluentWait se puede reutilizar en tantas esperas, con la misma latencia, como sea necesario.

```
Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
        .withTimeout(Duration.ofSeconds(30))
        .pollingEvery(Duration.ofSeconds(5))
        .ignoring(NoSuchElementException.class);

WebElement firstResult = wait.until(driver -> driver.findElement(By.xpath("//li/a")));
```

Espera implícita

- Una espera implícita es decirle a WebDriver que sondee el DOM durante un cierto período de tiempo al intentar encontrar un elemento o elementos si no están disponibles de inmediato. Esto puede ser útil cuando ciertos elementos en la página web no están disponibles de inmediato y necesitan algo de tiempo para cargarse.
- La espera implícita está deshabilitada de forma predeterminada y deberá habilitarse manualmente por sesión.

```
WebDriver driver = new FirefoxDriver();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
driver.get("http://somedomain/url_that_delays_loading");
WebElement element = driver.findElement(By.id("myDynamicElement"));
```

- Si se habilita la espera implícita se pierde la posibilidad de fijar esperas explícitas. No se debe mezclar esperas implícitas y explícitas. Mezclar esperas explícitas y esperas implícitas causará consecuencias no deseadas, es decir, esperara el máximo de tiempo incluso si el elemento está disponible o la condición es verdadera.

Mensajes emergentes

- WebDriver proporciona una API para trabajar con los tres tipos nativos de mensajes emergentes ofrecidos por JavaScript: alertas, prompts y confirmaciones. Estas ventanas emergentes están diseñadas por el navegador y ofrecen personalización limitada.
- **Alertas**: muestra un mensaje personalizado y un solo botón que cierra la alerta, etiquetado en la mayoría de los navegadores como OK.

```
Alert alert = wait.until(ExpectedConditions.alertIsPresent());
assertEquals(msgEsperado, alert.getText());
alert.accept(); // Cerrar
```

Mensajes emergentes

- **Confirm**: muestra un mensaje de confirmación y dos botones: uno para confirmar y otro para descartar.

```
Alert alert = wait.until(ExpectedConditions.alertIsPresent());
assertEquals(msgEsperado, alert.getText());
alert.accept();  // confirmar
alert.dismiss(); // descartar
```

- **Prompt**: solicita un texto al usuario, muestra un mensaje de petición, una caja de texto y dos botones: uno para aceptar y otro para cancelar.

```
Alert alert = wait.until(ExpectedConditions.alertIsPresent());
assertEquals(msgEsperado, alert.getText());
alert.sendKeys("Selenium");
alert.accept();  // aceptar
alert.dismiss(); // cancelar
```


Interacciones de usuario avanzadas

- WebDrive suministra los métodos de alto nivel sendKeys y click para simular las entradas habituales de teclado y ratón, pero dispone de la posibilidad de crear acciones compuestas con interacciones de menor nivel.
- La API de interacciones de usuario avanzadas es una API más completa para describir acciones que un usuario puede realizar en una página web. Esto incluye acciones como arrastrar y soltar o hacer clic en varios elementos mientras mantiene presionada la tecla Ctrl.
- La clase Actions implementa los patrones Builder y Composite para crea una acción compuesta que contiene todas las acciones especificadas por las llamadas al método. Para generar una secuencia de acciones, se usa el generador de acciones para construirla:

```
Actions provider = new Actions(driver);
Action action = provider.keyDown(Keys.CONTROL)
    .click(someElement).click(someOtherElement)
    .keyUp(Keys.CONTROL)
    .build();
```

Interacciones de usuario avanzadas

- El objetos Action solo permite la ejecución de las secuencia:
  action.perform();
- Si la secuencia es de un solo uso, se puede construir y ejecutar en un solo paso:

```
provider.keyDown(Keys.CONTROL)
    .click(someElement).click(someOtherElement)
    .keyUp(Keys.CONTROL)
    .perform();
```

- La secuencia de acciones debe ser corta: es mejor realizar una secuencia corta de acciones y verificar que la página esté en el estado correcto antes de que tenga lugar el resto de la secuencia.

Teclado

- Hasta ahora, la interacción del teclado se realizaba a través de un elemento específico y WebDriver se aseguraba de que el elemento estuviera en el estado adecuado para esta interacción. Esto consistía principalmente en desplazar el elemento a la ventana gráfica y centrarse en el elemento.
- Dado que el API de interacciones adopta un enfoque orientado al usuario, es más lógico interactuar explícitamente con el elemento antes de enviarle texto, como lo haría un usuario. Esto significa hacer clic en un elemento o enviar un mensaje Keys.TAB cuando se enfoca en un elemento adyacente.
- En el API de interacciones se define la secuencia de acciones de teclado sin proporcionar un elemento, se aplican al elemento actual que tiene el foco. Posteriormente se puede cambiar el foco a un elemento antes de enviarle eventos de teclado.

```
Actions provider = new Actions(driver);
Action action = provider.action.keyDown(Keys.SHIFT).sendKeys(search,"Hola
        ").keyUp(Keys.SHIFT).sendKeys("Mundo").sendKeys(Keys.TAB).build();
action.perform();
```

Ratón

- Las acciones del mouse tienen un contexto: la ubicación actual del mouse. Entonces, al establecer un contexto para varias acciones del mouse (usando onElement), la primera acción será relativa a la ubicación del elemento utilizado como contexto, la siguiente acción será relativa a la ubicación del mouse al final de la última acción, etc.
- Mientras que el método de alto nivel sendKeys cubre la mayoría de los escenarios de teclado y no suele requerir acciones de bajo nivel, el método click cubre el escenario mas común para el ratos pero deja fuera operaciones habituales como el doble click, los desplazamientos, el menú contextual, arrastrar y soltar, … Para todas estas acciones es necesario el API de interacciones.

```
Actions provider = new Actions(driver);

provider.clickAndHold(sourceEle).moveToElement(targetEle) .release().build().perform();
```


Acciones

- Teclado:
  - SendKeysAction: equivalente a WebElement.sendKey(...)
  - KeyDownAction: mantener presionada una tecla modificadora.
  - KeyUpAction: liberar la tecla modificadora.

- Ratón:
  - ClickAction: equivalente a WebElement.click()
  - DoubleClickAction: hacer doble clic en un elemento.
  - ClickAndHoldAction: mantenga presionado el botón izquierdo del mouse.
  - ButtonReleaseAction: liberar un botón del ratón retenido.
  - ContextClickAction: hacer clic con el botón secundario del ratón, (que generalmente abre el menú contextual).
  - MoveMouseAction: mover el ratón de su ubicación actual a otro elemento.
  - MoveToOffsetAction: mover el ratón una cantidad (desplazamiento) desde un elemento (el desplazamiento podría ser negativo y el elemento podría ser el mismo elemento al que se acaba de mover el ratón).

Trabajando con elementos select

- A la hora de seleccionar elementos puede ser necesario código repetitivo para poder ser automatizado.
- Para reducir esto y hacer tus test mas limpios, existe un clase Select en los paquetes de soporte de Selenium.

```
import org.openqa.selenium.support.ui.Select;
```

- El WebElement debe ser envuelto por un objeto Select para acceder a la funcionalidad extendida:

```
Select selectObject = new Select(driver.findElement(By.id("selectElementID")));
```

- La funcionalidad extendida permite:

```
selectObject.selectByIndex(1);
selectObject.selectByValue("value1");
selectObject.selectByVisibleText("Bread");
WebElement firstSelectedOption = selectObject.getFirstSelectedOption();
List<WebElement> allSelectedOptions = selectObject.getAllSelectedOptions();
selectObject.deselectByIndex(1);
selectObject.deselectByValue("value1");
selectObject.deselectByVisibleText("Bread");
selectObject.deselectAll();
```


Trabajando con colores

- En algunas ocasiones es posible que sea necesario querer validar el color de algo como parte de las pruebas. El problema es que las definiciones de color en la web no son constantes.
- La clase org.openqa.selenium.support.Color es una forma sencilla de comparar una representación de color HEX con una representación de color RGB, o una representación de color RGBA con una representación de color HSLA.

```
private final Color HEX_COLOUR = Color.fromString("#2F7ED8");
private final Color RGB_COLOUR = Color.fromString("rgb(255, 255, 255)");
private final Color RGB_COLOUR = Color.fromString("rgb(40%, 20%, 40%)");
private final Color RGBA_COLOUR = Color.fromString("rgba(255, 255, 255, 0.5)");
private final Color RGBA_COLOUR = Color.fromString("rgba(40%, 20%, 40%, 0.5)");
private final Color HSL_COLOUR = Color.fromString("hsl(100, 0%, 50%)");
private final Color HSLA_COLOUR = Color.fromString("hsla(100, 0%, 50%, 0.5)");
private final Color BLACK = Color.fromString("black");
private final Color TRANSPARENT = Color.fromString("transparent");
```

- Para crear las aserciones:

```
Color color = Color.fromString(driver.findElement(By.id("login")).getCssValue("color"));
assertEquals(TRANSPARENT, color);
assertEquals("#2F7ED8", color.asHex());
```

Excepciones

- NoSuchElementException: Se produce cuando se intenta interactuar con un elemento que no satisface la estrategia de selector.

```
driver.findElement(By.linkText("Este no existe")).click();
```

- StaleElementReferenceException: Se produce cuando un elemento que se identificó y almaceno previamente ha sido modificado en el DOM y se intentó interactuar con él después de la mutación.

```
element = wait.until(driver -> driver.findElement(By.id("txtSaluda")));
driver.findElement(By.linkText("Next")).click();
element.sendKeys("oHola");
```

- ElementNotInteractableException: Lanzada para indicar que aunque un elemento está presente en el DOM, no está en un estado con el que se pueda interactuar: fuera de la vista (scroll, solapado, oculto …).

```
Actions provider = new Actions(driver);
provider.sendKeys(Keys.PAGE_DOWN).perform();
provider.keyDown(Keys.CONTROL).sendKeys(Keys.END) .keyUp(Keys.CONTROL).perform();
```


RECOMENDACIONES

Patrones de diseño

- Page Objects: una simple abstracción de la interfaz de usuario de su aplicación web.
- Componente Loadable: Modelado de PageObjects como componentes.
- BotStyleTests: Uso de un enfoque basado en comandos para automatizar pruebas, en lugar del enfoque basado en objetos que PageObjects fomenta
- Domain Driven Design: Exprese las pruebas en el idioma del usuario final de la aplicación.
- Acceptance Test Driven Development (ATDD): técnica conocida también como Story Test-Driven Development (STDD), utiliza las pruebas de aceptación para ayudar a estructurar el trabajo de desarrollo.

Lenguaje de dominio específico

- Un lenguaje de dominio específico (DSL) es un sistema que proporciona al usuario un medio expresivo para resolver un problema. Permite a un usuario interactuar con el sistema en sus términos, no solo en jerga del programador.
- A los usuarios, en general, no les importa principalmente cómo se ve su sitio, no están preocupados por la decoración, animaciones o gráficos. Lo que realmente les importa es como el sistema se impulsa a través de los procesos con una mínima dificultad. El trabajo del probador debe acercarse lo más que pueda a “capturar” esta mentalidad utilizando un lenguaje ubicuo (la terminología del usuario). Con eso en mente, los scripts de prueba deberían se legibles al usuario.
- Con Selenium, el DSL generalmente se representa por métodos, la elección de los sus nombres es de suma importancia, así como la refactorización para ocultar la complejidad.
- Beneficios
  - Legible: Las partes interesadas del negocio pueden entenderlo.
  - Escribible: Fácil de escribir, evita duplicaciones innecesarias.
  - Extensible: Se puede agregar funcionalidad (razonablemente) sin romper los contratos y la funcionalidad existente.
  - Mantenible: Al dejar los detalles de implementación fuera de casos de prueba, está bien aislado contra cambios en el aplicación.

Patrón Page Objects

- https://martinfowler.com/bliki/PageObject.html
- Cuando se escriben pruebas de una página web, hay que acceder a los elementos dentro de esa página web para hacer clic en los elementos, teclear entradas y determinar lo que se muestra.
- Sin embargo, si se escriben pruebas que manipulan los elementos HTML directamente, las pruebas serán frágiles ante los cambios en la interfaz de usuario.
- Un objeto de página envuelve una página HTML, o un fragmento, con una API específica de la aplicación, lo que permite manipular los elementos de la página sin excavar en el HTML.

Patrón Page Objects

- La regla básica para un objeto de página es que debe permitir que un cliente de software haga cualquier cosa y vea todo lo que un humano puede hacer.
- El objeto de página debe proporcionar una interfaz que sea fácil de programar y oculta en la ventana.
- Entonces, para acceder a un campo de texto, debe tener métodos de acceso que tomen y devuelvan una cadena, las casillas de verificación deben usar valores booleanos y los botones deben estar representados por nombres de métodos orientados a la acción.
- El objeto de la página debe encapsular los mecanismos necesarios para encontrar y manipular los datos en el propio control GUI.
- A pesar del término objeto de "página", estos objetos no deberían construirse para cada página, sino para los elementos significativos en una página.
- Los problemas de concurrencia y asincronía son otro tema que un objeto de página puede encapsular.

Ventajas del patrón Page Objects

- De acuerdo con patrón Page Object, deberíamos mantener nuestras pruebas y localizadores de elementos por separado, esto mantendrá el código limpio y fácil de entender y mantener.
- El enfoque Page Object hace que el programador de marcos de automatización de pruebas sea más fácil, duradero y completo.
- Otra ventaja importante es que nuestro repositorio de objetos de página es independiente de las pruebas de automatización. Mantener un repositorio separado para los objetos de la página nos ayuda a usar este repositorio para diferentes propósitos con diferentes marcos como, podemos integrar este repositorio con otras herramientas como JUnit / NUnit / PhpUnit, así como con TestNG / Cucumber / etc.
- Los casos de prueba se vuelven cortos y optimizados, ya que podemos reutilizar los métodos de objetos de página.
- Los casos de prueba se centran solamente en el comportamiento.
- Cualquier cambio en la IU se puede implementar, actualizar y mantener fácilmente en los objetos y clases de página sin afectar a los casos de pruebas que no estén implicados.

Page Objects

```
public class LoginPage {
    private WebDriver driver;
    private final By txtUsuarioBy = By.id("txtUsuario");
    private final By txtPasswordBy = By.id("txtPassword");
    private final By btnSendLoginBy = By.id("btnSendLogin");
    private final By userDataBy = By.id("userData");

    public LoginPage(WebDriver driver) {
        this.driver = driver;
        driver.get("http://localhost/login");
    }
    public void ponUsuario(String valor) { driver.findElement(txtUsuarioBy).sendKeys(valor); }
    public void ponPassword(String valor) { driver.findElement(txtPasswordBy).sendKeys(valor); }
    public void enviarLogin() { driver.findElement(btnSendLoginBy).click();          }
    public String textoSaludo() {
        WebElement element = new WebDriverWait(driver, Duration.ofSeconds(3).getSeconds()).until(driver ->
            driver.findElement(userDataBy));
        return element.getText();
    }
}
```

Prueba usando un Page Objects

```
@Test
public void loginTest() {
    LoginPage page = new LoginPage(driver);
    page.ponUsuario("admin");
    page.ponPassword("P@$$w0rd");
    page.enviarLogin();
    assertThat(page.textoSaludo(), is("Hola Administrador"));

}
```


PageFactory

- La biblioteca de soporte de WebDriver contiene una clase de PageFactory que simplifica la implementación del patón PageObject.
- La PageObject declara sus campos como WebElements o List <WebElement>:
  private WebElement txtUsuario;
- Cuando ejecutamos la factoría, PageFactory buscará un elemento en la página que coincida con el nombre del campo WebElement en la clase e inyectara el elemento en el campo. Para ello, primero busca un elemento con un atributo de ID coincidente. Si esto falla, recurre a la búsqueda de un elemento por el valor de su atributo "name".
  LoginPageFactory page = new LoginPageFactory(driver);
  PageFactory.initElements(driver, page);
- La anotación @FindBy permite elegir un nombre significativo de campo y cambiar la estrategia utilizada para buscar el elemento:
  @FindBy(how = How.CSS, css = "#userData")
  private WebElement saludo;

PageFactory

```
public class LoginPageFactory {
        private WebDriver driver;
        private WebElement txtUsuario;
        private WebElement txtPassword;
        private WebElement btnSendLogin;
        @FindBy(how = How.CSS, css = "#userData")
        private WebElement saludo;

        public LoginPageFactory(WebDriver driver) { this.driver = driver; }
        public void ponUsuario(String valor) { txtUsuario.sendKeys(valor); }
        public void ponPassword(String valor) { txtPassword.sendKeys(valor); }
        public void enviarLogin() { btnSendLogin.click(); }
        public String textoSaludo() { return saludo.getText(); }
}

LoginPageFactory page = PageFactory.initElements(driver, LoginPageFactory.class);
```


Pruebas de estilo bot

- Un "bot" es una abstracción orientada a la acción sobre las API de Selenium sin procesar. Permite ocultar la complejidad del API con comandos mayor nivel que son fácil de cambiar.

```
public class ActionBot {
  private final WebDriver driver;
  public ActionBot(WebDriver driver) { this.driver = driver; }
  public void click(By locator) { driver.findElement(locator).click(); }
  public void submit(By locator) { driver.findElement(locator).submit(); }
  public void type(By locator, String text) {
    WebElement element = driver.findElement(locator);
    element.clear();
    element.sendKeys(text);
  }
 }
```

- Una vez que se han construido estas abstracciones y se ha identificado la duplicación en sus pruebas, es posible colocar los PageObjects encima de los bots.

API fluída

- Un API fluida, termino acuñado por Martin Fowler y Eric Evans, permite encadenar varias acciones consecutivas proporcionando una sensación más fluida al código.

```
String saludo = sitio.pedirLogin().rellenarFormulario("admin",
        "P@$$w0rd").enviarFormulario().dameTextoSaludo();
```

- Crear un API fluida significa construirla de tal manera que cumpla con los siguientes criterios:
  - El usuario de la API puede entenderla API muy fácilmente.
  - El API puede realizar una serie de acciones para finalizar una tarea.
  - El nombre de cada método debe utilizar la terminología específica del dominio.
  - La API debe ser lo suficientemente sugerente como para guiar a los usuarios del API sobre qué hacer a continuación y qué posibles operaciones se pueden en un determinado momento.
- Son interfaces o clases que cuando invocamos a un método concreto nos devuelve el mismo objeto modificado u otro objeto. De tal forma que podemos volver a solicitar otro método del mismo objeto, y encadenar más operaciones, o continuar las operaciones con el nuevo objeto.
- Se recomienda considerar el uso del patrón de diseño Fluent API en el objeto de página. El API de Seleniun WebDriver suministra la clase base LoadableComponent que tiene como objetivo simplificar la creación de PageObjects fluidos.

LoadableComponent

- LoadableComponent proporciona una forma estándar de garantizar que las páginas se carguen y facilita la depuración de la errores de carga de la página. Ayuda a reducir la cantidad de código repetitivo en la pruebas, lo que a su vez hace que el mantenimiento de la pruebas sea menos agotador.
- La clase cuenta con tres métodos:
  - public T get():
    - Devuelve el PageObject si la página esta cargada o cargándola si es necesario.
  - protected abstract void load():
    - Cuando este método termine, el componente modelado por la subclase debería estar completamente cargado. Se espera que esta subclase navegue a una página apropiada si fuera necesario.
  - protected abstract void isLoaded() throws Error
    - Determine si el componente está cargado o no, lanzando un Error (no una excepción) cuando no este cargada u obsoleta. se carga el componente, este método volverá, pero cuando no se carga, se debe lanzar un. Esto permite la verificación compleja y el informe de errores al cargar una página o cuando una página no se carga.

LoadableComponent

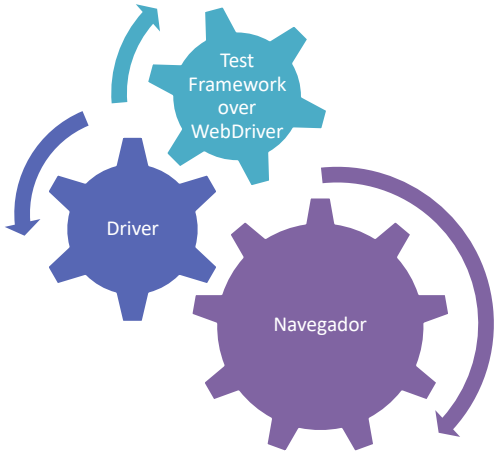
```
public class LoginPageFluent {
        private WebDriver driver;
        private WebElement txtUsuario;
        private WebElement txtPassword;
        private WebElement btnSendLogin;
        @FindBy(how = How.CSS, css = "#userData")
        private WebElement saludo;

        public LoginPageFactory(WebDriver driver) { this.driver = driver; }
        @Override
        protected void load() {
                driver.get("http://localhost/login");
                PageFactory.initElements(driver, this);
        }
        @Override
        protected void isLoaded() throws Error {
                if(driver.getTitle() == null || !driver.getTitle().contains("Login"))
                        throw new Error();
        }
```

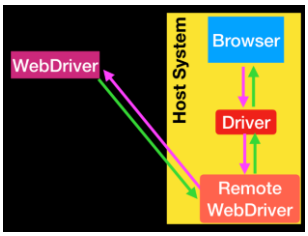

LoadableComponent

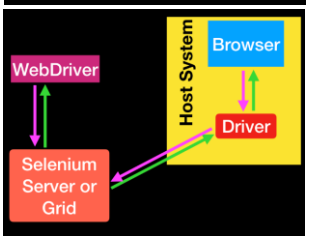
```
        public LoginPageFluent ponUsuario(String valor) {
            txtUsuario.sendKeys(valor);  return this;
        }
        public LoginPageFluent ponPassword(String valor) {
            txtPassword.sendKeys(valor); return this;
        }
        public LoginPageFluent enviarLogin() {
            btnSendLogin.click(); return this;
        }
        public String textoSaludo() { return saludo.getText(); }
}

LoginPageFluent page = new LoginPageFluent();
assertEquals("Hola Administrador", page.ponUsuario("admin")
            .ponPassword("P@$$w0rd").enviarLogin().textoSaludo());
```

SELENIUM GRID

Interacciones







Remote WebDriver

- Se puede usar WebDriver de forma remota de la misma manera que se usaría localmente. La principal diferencia es que un WebDriver remoto debe ser configurado para que pueda ejecutar las pruebas en una máquina diferente.
- Un WebDriver remoto se compone de dos piezas: un cliente y un servidor. El cliente es la prueba de WebDriver y el servidor es simplemente un servlet Java, que se puede alojar en cualquier servidor moderno de aplicaciones JEE.
- Pros
  - Separa dónde se ejecutan las pruebas desde donde está el navegador.
  - Permite que las pruebas se ejecuten con navegadores no disponibles en el sistema operativo actual
- Contras
  - Requiere que se ejecute un contenedor de servlet externo
  - Puede encontrar problemas con los finales de línea al obtener texto del servidor remoto
  - Introduce latencia adicional a las pruebas, sobre todo con las excepciones.

Servidor

- El servidor siempre se ejecutará en la máquina con el navegador que deseas probar. Debe tener instalados los diferentes driver para los navegadores.
- El servidor se puede usar desde la línea de comandos o mediante configuración de código.

```
java -jar selenium-server-standalone-{VERSION}.jar
```

- Para personalizar la configuración por defecto, se utiliza un archivo de configuración JSON:

```
{
  "port": 4444, "browserTimeout": 0, "timeout": 1800,
  "debug": false, "enablePassThrough": true
}
```

```
java -jar selenium-server-standalone.jar -config standaloneConfig.json
```

- Esta disponible una consola en: http://localhost:4444/

Cliente

- Las pruebas que quieran ejecutarse en remoto deben sustituir la instancia de WebDriver por una instancia de RemoteWebDriver.
- El constructor recibe dos parámetros:
  - URL con la dirección del servidor que ejecutara las pruebas.
  - Una instancia de las opciones del navegador que debe usar el servidor remoto (browserName).
- Para implementar la prueba de ejecución remota:

```
FirefoxOptions firefoxOptions = new FirefoxOptions();
ChromeOptions chromeOptions = new ChromeOptions();

WebDriver driver = new RemoteWebDriver(new URL("http://localhost:4444/wd/hub"),
chromeOptions);
driver.get("http://www.google.com");
driver.quit();
```


Grid

- Selenium Grid es un servidor inteligente que actúa de proxy lo que permite a los tests de Selenium enrutar sus comandos hacia instancias remotas de navegadores web. La intención es proporcionar una forma sencilla de ejecutar los tests en paralelo en múltiple maquinas.
- Con Selenium Grid un servidor actúa como el centro de actividad (hub) encargado de enrutar los comandos de los tests en formato JSON hacia uno o mas nodos registrados en el Grid. Los tests contactan con el hub para obtener acceso a las instancias remotas de los navegadores.
- Selenium Grid permite ejecutar los tests en paralelo en múltiples maquinas y gestionar diferentes versiones de navegadores y configuraciones de manera centralizada (en lugar de hacerlo de manera individual en cada test).

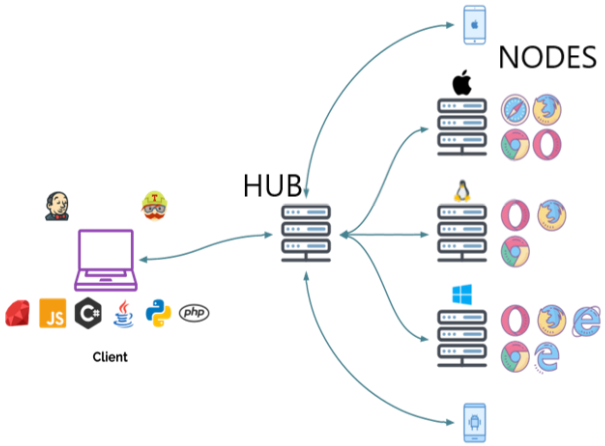
Grid

- Selenium Grid no es una solución mágica para todos los problemas. Permite resolver un subconjunto de problemas comunes de delegación y distribución, pero no administrará su infraestructura y podría no satisfacer necesidades concretas.
- Sus principales características son:
  - Punto de entrada centralizado para todos los tests
  - Gestión y control de los nodos / entornos donde se ejecutan los navegadores
  - Escalado
  - Balanceo de carga
  - Ejecución de los tests en paralelo
  - Testing cruzado entre diferentes sistemas operativos

Casos de uso

- El Grid se usa para acelerar la ejecución de los test usando múltiples maquinas para ejecutarlos en paralelo: en cuatro máquinas tardaría aproximadamente una cuarta parte de lo que tardaría en una sola. Esto permite ejecutar suites muy grandes, y de larga duración (horas), en un tiempo razonable, así como obtener una retroalimentación temprana.
- El Grid permite ejecutar la suite múltiples entornos, especialmente, contra diferentes navegadores y versiones al mismo tiempo, lo cual seria imposible en una sola maquina. Cuando la suite de test es ejecutada, el Selenium Grid recibe cada combinación de test-navegador y lo asigna para su ejecución a la maquina o maquinas con el navegador requerido.
- Remote WebDriver trabaja en escenarios de uso con un cliente para un remoto, Selenium Grid reutiliza la misma infraestructura para que un cliente use múltiples remotos.

Arquitectura



Hub

- El Hub es un punto central donde se envían todos los tests. Cada Grid tiene un único hub. El hub necesita ser accesible desde la perspectiva de los clientes (ej. Servidor de la CI, maquina del desarrollador). El hub se conectará a uno o mas nodos en los que los tests serán ejecutados.
- Características:
  - Ejerce como mediador y administrador
  - Acepta peticiones para ejecutar los tests
  - Recoge instrucciones de los clientes y las ejecuta de forma remota en los nodos
  - Gestiona los hilos

Hub

- Para arrancar el Hub:
  java -jar selenium-server-standalone.jar -role hub *-hubConfig hubConfig.json -debug*
- Para personalizar la configuración por defecto, se utiliza un archivo de configuración JSON:

```
{
  "port": 4444, "browserTimeout": 0, "timeout": 1800, "debug": false,
  "newSessionWaitTimeout": -1, "cleanUpCycle": 5000,
  "throwOnCapabilityNotPresent": true,
  "servlets": [], "withoutServlets": [], "custom": {},
  "capabilityMatcher": "org.openqa.grid.internal.utils.DefaultCapabilityMatcher",
  "registry": "org.openqa.grid.internal.DefaultGridRegistry"
}
```

- El hub escuchará al puerto 4444 por defecto. Se puede ver el estado del hub abriendo una ventana del navegador y navegando a http://localhost:4444/grid/console.

Nodos

- Los nodos son diferentes instancias de Selenium que ejecutarán los tests en sistemas informáticos individuales. Puede haber tantos nodos en un grid como sean necesarios.
- Las maquinas que contienen los nodos no necesitan disponer del mismo sistema operativo o el mismo conjunto de navegadores que el hub o los otros nodos. Un nodo en Windows podría tener la capacidad de ofrecer Internet Explorer como opción del navegador mientras que esto no podría ser posible en Linux o Mac.
- Características:
  - Donde se ubican los navegadores
  - Se registra a si mismo en el hub y le comunica sus capacidades
  - Recibe las peticiones desde el hub y las ejecuta

Nodos

- Al iniciar los nodos deben indicar la url del hub (si no se especifica un puerto a través del parámetro -port se elegirá un puerto libre):

  java -jar selenium-server-standalone.jar -role node -hub http://localhost:4444
- Para personalizar la configuración por defecto, se utiliza un archivo de configuración JSON:

```
{
  "capabilities": [
    { "browserName": "firefox", "maxInstances": 2, "seleniumProtocol": "WebDriver" },
    { "browserName": "chrome", "maxInstances": 3, "seleniumProtocol": "WebDriver" }
  ],
  "proxy": "org.openqa.grid.selenium.proxy.DefaultRemoteProxy",
  "maxSession": 5, "port": -1, "register": true, "registerCycle": 5000,
  "hub": "http://localhost:4444", "nodeStatusCheckTimeout": 5000, "nodePolling": 5000,
  "unregisterIfStillDownAfter": 60000, "downPollingLimit": 2, "debug": false,
  "servlets": [], "withoutServlets": [], "custom": {}
}
```

```
java -jar selenium-server-standalone.jar -role node -nodeConfig nodeConfig.json
```


Depuración

- Se pueden ejecutar múltiples hub y nodos en la misma maquina, con el parámetro -port se especifica un puerto diferente, pero tiene poca utilidad dado que comparten recursos y configuraciones, invalidando los objetivos de multicidad del Grid.
- Para resolución avanzada de problemas, se puede especificar un archivo de log que almacene los mensajes del sistema con el argumento -log:

  java -jar selenium-server-standalone.jar -role hub -log log.txt
- Se puede usar el argumento -debug para imprimir los logs de depuración en la consola:

  java -jar selenium-server-standalone.jar -role node -debug

Seguridad

- El Grid de Selenium debe estar protegido contra accesos externos mediante el uso apropiado de los permisos del firewall.
- Fallar a la hora de proteger el Grid puede ocasionar uno o mas de los siguientes problemas:
  - Permitir acceso abierto a tu infraestructura del Grid.
  - Permitir a terceros el acceso a aplicaciones web y archivos interno.
  - Permitir a terceros ejecutar tus ejecutables.

Cloud

- Para usar Selenium Grid, se necesita mantener una infraestructura para los nodos. Como esto puede suponer un engorro y un gran esfuerzo de tiempo y recursos, se pueden usar proveedores de IaaS (Infraestructura como servicio) en la nube como Amazon EC2 o Google Compute para proveer de la infraestructura. Otras opciones incluyen usar proveedores como Sauce Labs or Testing Bot los cuales proveen Selenium Grid como servicio en la nueve.
- Docker provee una forma conveniente de aprovisionar y escalar la infraestructura de Selenium Grid en unidades conocidas como contenedores. Los contenedores son unidades estandarizadas de software que contienen todo lo necesario para ejecutar la aplicación deseada, incluidas todas las dependencias, en un entorno confiable y regenerable en diferentes sistemas.
- El proyecto de Selenium mantiene un conjunto de imágenes Docker las cuales se pueden descargar y ejecutar para tener un Grid funcionando rápidamente. Los nodos están disponibles para los navegadores Firefox y Chrome.

https://www.getpostman.com/

POSTMAN

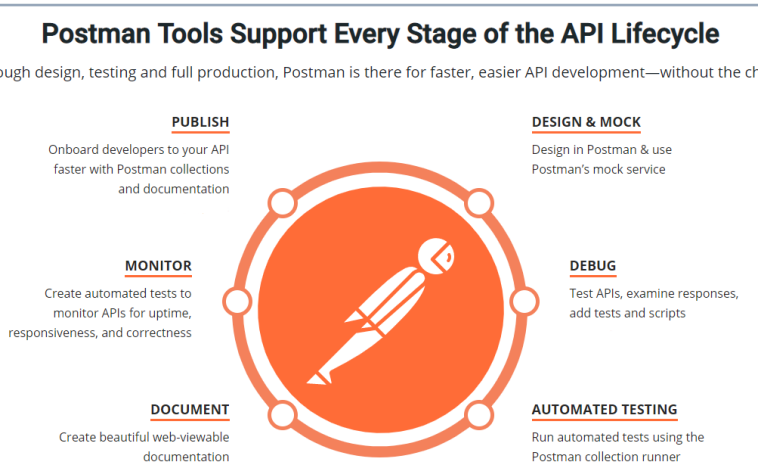
Pruebas de la API

- Ejercitar todas las rutas y parámetros para comprobar que invocan las operaciones correctas.
- Verificar que cada operación devuelve los códigos de estado HTTP correctos para diferentes combinaciones de entradas.
- Comprobar que todas las rutas estén protegidas correctamente y que estén sujetas a las comprobaciones de autenticación y autorización apropiadas.
- Verificar el control de excepciones que realiza cada operación y que se devuelve una respuesta HTTP adecuada y significativa de vuelta a la aplicación cliente.
- Comprobar que los mensajes de solicitud y respuesta están formados correctamente.
- Comprobar que todos los vínculos dentro de los mensajes de respuesta no están rotos.

Introducción

- Postman surgió originariamente como una extensión para el navegador Google Chrome que permitía realizar peticiones API REST con métodos diferentes al GET. A día de hoy dispone de aplicaciones nativas para MAC, Windows y algunas producciones Linux.
- Está compuesto por diferentes herramientas y utilidades gratuitas (en la versión free) que permiten realizar tareas diferentes dentro del mundo API REST: creación de peticiones a APIs internas o de terceros, elaboración de tests para validar el comportamiento de APIs, posibilidad de crear entornos de trabajo diferentes (con variables globales y locales), y todo ello con la posibilidad de ser compartido con otros compañeros del equipo de manera gratuita (exportación de toda esta información mediante URL en formato JSON).
- Además, dispone de un modo cloud colaborativo (de pago) para que equipos de trabajo puedan desarrollar entre todos colecciones para APIs sincronizadas en la nube para una integración más inmediata y sincronizada.
- Quizás sea una de las herramientas más utilizadas para hacer testing exploratorio de API REST.

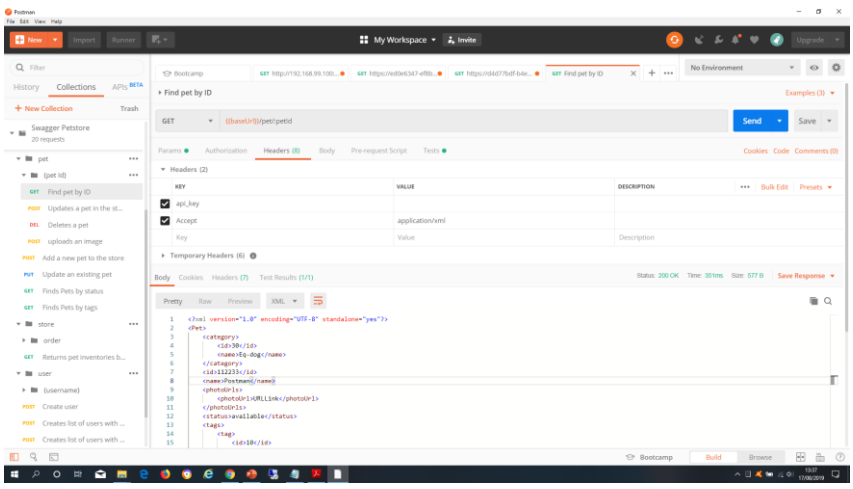
Soporte del ciclo de vida



Instalación

- Postman está disponible como una aplicación nativa para los sistemas operativos macOS, Windows (32 bits y 64 bits) y Linux (32 bits y 64 bits).
- La extensión para Chrome de Postman ha quedado en desuso.
- Para descargar Postman accedemos a su página oficial:
  - https://www.getpostman.com/downloads/
- Tras la instalación se nos solicita introducir una cuenta que tengamos ya registrada o bien se nos ofrece la posibilidad de crear una nueva cuenta gratuita.
- Esto se debe a que Postman tiene muchas funciones que interactúan con una nube, por lo que para poder almacenar un registro de las peticiones y de nuestro trabajo en la nube, y poder trabajar compartiendo un workspace o espacio de trabajo con otros compañeros de equipo, necesitamos identificarnos con una cuenta de Postman.

Interfaz de Postman



Espacios de trabajo

- Un espacio de trabajo es un repositorio de todos los artefactos de Postman que pueden ser persistentes: colecciones, entornos, simulacros, monitores y otros.
- Las personas pueden organizar su trabajo en espacios de trabajo personales y los equipos pueden colaborar en espacios de trabajo en equipo.
- Independientemente del tipo de espacio de trabajo, se puede compartir elementos en múltiples espacios de trabajo al mismo tiempo.
- Si no se tiene una cuenta de Postman, solo se puede tener un espacio de trabajo personal. Si se dispone de una cuenta, se pueden crear un número ilimitado de espacios de trabajo.
- Los espacios de trabajo de equipo (disponibles solo con usuarios de Postman) permiten el desarrollo continuo o colaboraciones a corto plazo. Cada equipo (sin coste un máximo de 25 usuarios) tiene un espacio de trabajo de equipo predeterminado y puede crear un número ilimitado de espacios de trabajo. Se puede compartir una colección con otros espacios de trabajo. Una Colección Postman compartida con los miembros del equipo es visible para todos los miembros de su equipo Postman con un acceso predeterminado de solo lectura. Si se tiene los permisos adecuados para la colección, se puede editar y actualizar en cualquier espacio de trabajo donde exista, permitiendo el seguimiento de cambios.

Espacios de trabajo

- Organización personal:
  - Si un espacio de trabajo personal está abarrotado, se puede crear un nuevo espacio de trabajo para reflejar su flujo de trabajo personal de una manera más granular.
- Organización del equipo:
  - Del mismo modo, cree nuevos espacios de trabajo para reflejar el flujo de trabajo de su equipo.
  - Estos pueden ser espacios de trabajo permanentes alineados por función, producto, proyecto o socio.
  - Estos también pueden ser espacios de trabajo temporales para proyectos y actividades temporales.
- Fuente de la verdad:
  - Los equipos necesitan una única fuente de verdad para sus API, y los espacios de trabajo pueden dar confianza a un equipo o sub-equipo de que están trabajando con las últimas versiones de las colecciones Postman y saber que sus cambios se están sincronizando en tiempo real.
- Permisos del equipo:
  - Los espacios de trabajo del equipo son de solo lectura de forma predeterminada.
  - Se pueden controlar los permisos de edición y visualización de las colecciones y todos sus elementos asociados al proporcionar capacidades de edición a todo el equipo o solo a individuos específicos del equipo.

Espacios de trabajo

- Descubrimiento:
  - Es un escenario común que alguien estará trabajando en algo de lo que el equipo más amplio no sabe nada.
  - Si hay colecciones compartidas en un espacio de trabajo común, esto permite a todos los miembros del equipo comprender el alcance de un proyecto y ver esos elementos con mayor claridad.
- Feed de actividad actualizado:
  - Ver una fuente de actividad de operaciones CUD (Crear, Actualizar, Eliminar) dentro de una colección.
  - Mantiene al tanto de quién actualizó una colección, cuáles fueron las actualizaciones y cuándo se completaron.
- Depuración en tiempo real con Historial:
  - Cuando los miembros del equipo se unen a un espacio de trabajo, incluso el historial de solicitudes es una entidad compartida.
  - Todos los que forman parte de ese espacio de trabajo pueden ver las solicitudes enviadas más recientemente y observar el comportamiento en sus propias instancias en tiempo real.

Colecciones

- Las colecciones Postman permiten agrupar solicitudes individuales y organizar estas solicitudes en carpetas.
- Organización: Se puede guardar las solicitudes en colecciones y carpetas, para no tener que buscar en el historial repetidamente.
- Documentación: Se puede agregar un nombre y descripciones a solicitudes, carpetas y colecciones. En Postman, se puede usar el navegador de colección para ver la documentación o generar y publicar páginas con la documentación del API.
- Suites de prueba: Se pueden adjuntar scripts de prueba a las solicitudes y crear conjuntos de pruebas de integración.
- Flujos de trabajo condicionales: Se pueden usar scripts para pasar datos entre solicitudes de API y crear flujos de trabajo que reflejen su caso de uso de API real.
- Compartir: Se pueden compartir colecciones directamente a través de espacios de trabajo o mediante la exportación a ficheros.
- Las colecciones son los puntos de partida para las simulaciones (Service Mocking), la documentación, la monitorización y otros.

Variables y Entornos

- Postman permite sustituir determinados valores constantes por el valor almacenado en una variable (o propiedad). Las variables le permiten:
  - Reutilizar los valores para mantener el código SECO (DRY: Don't Repeat Yourself).
  - Configurar diferentes juegos de valores para diferentes entornos.
  - Extraer datos de respuestas y solicitudes en cadena en una colección.
- Las variables son nombres simbólicos que representan la información que almacena en ellas. La información que representan las variables puede cambiar, pero las operaciones con la variable siguen siendo las mismas. Las variables en Postman funcionan de la misma manera.
- Un entorno es un conjunto de pares clave-valor. La clave representa el nombre de la variable. Mientras se trabaja con las API, a menudo se necesita diferentes configuraciones para la máquina local, el servidor de desarrollo o la API de producción. Los entornos permiten personalizar las solicitudes utilizando variables para cambiar entre diferentes configuraciones sin cambiar las solicitudes.
- Las variables globales proporcionan un conjunto de variables que siempre están disponibles en todos los ámbitos. Se pueden tener múltiples entornos, pero solo uno puede estar activo a la vez con un conjunto de variables globales, que siempre están disponibles.

Variables

- Las variables se pueden definir a diferentes niveles que determinan el ámbito y alcance (si una variable comparte su nombre con una variable superior, la variable mas local tendrá prioridad).
  1. Global
  2. Colección
  3. Entorno
  4. Datos
  5. Local
- Las variables globales y de entorno se gestionan mediante el "Manage Environments", las de colección mediante la edición de la colección, las de datos cargando el fichero (CSV o JSON) en el Collection Runner y las locales en los script.
- Para acceder a variables en el generador de solicitudes, la cadena {{variableName}} se reemplazará con su valor actual cuando Postman resuelva la variable.
- En los script, las variables locales son accesibles de la forma habitual. Para acceder a niveles superiores es necesario indicar el nivel (globals, variables, environment, iterationData, cookies) y el método: pm.nivel.get() o pm.nivel.set().
- Salvo las variables locales, los valores siempre se almacenan como cadenas. Si se está almacenando objetos o matrices, es necesario serializarlos a cadena JSON con JSON.stringify() o des serializarlos con JSON.parse().

Variables dinámicas

- Postman tiene algunas variables dinámicas que puede usar en las solicitudes. Las variables dinámicas no se pueden usar en los script, solo son accesible por el generador de solicitudes en la URL, encabezados y cuerpo.

Comunes: $guid, $timestamp, $randomUUID

Texto, números y colores: $randomAlphaNumeric, $randomBoolean, $randomInt, $randomColor, $randomHexColor, $randomAbbreviation

Fechas: $randomDateFuture, $randomDatePast, $randomDateRecent, $randomWeekday, $randomMonth

Archivos y directorios: $randomFileName, $randomFileType, $randomFileExt, $randomCommonFileName, $randomCommonFileType, $randomCommonFileExt, $randomFilePath, $randomDirectoryPath, $randomMimeType

Imágenes: $randomImage, $randomAvatarImage, $randomImageUrl, $randomAbstractImage, $randomAnimalsImage, $randomBusinessImage, $randomCatsImage, $randomCityImage, $randomFoodImage, $randomNightlifeImage, $randomFashionImage, $randomPeopleImage, $randomNatureImage, $randomSportsImage, $randomTechnicsImage, $randomTransportImage, $randomImageDataUri

Bases de datos: $randomDatabaseColumn, $randomDatabaseType, $randomDatabaseCollation, $randomDatabaseEngine

Internet y direcciones IP: $randomIP, $randomIPV6, $randomMACAddress, $randomPassword, $randomLocale, $randomUserAgent, $randomProtocol, $randomSemver

Variables dinámicas

Dominios, correos electrónicos y nombres de usuario: $randomDomainName, $randomDomainSuffix, $randomDomainWord, $randomEmail, $randomExampleEmail, $randomUserName, $randomUrl

Nombres: $randomFirstName, $randomLastName, $randomFullName, $randomNamePrefix, $randomNameSuffix

Profesión: $randomJobArea, $randomJobDescriptor, $randomJobTitle, $randomJobType

Teléfono, dirección y ubicación: $randomPhoneNumber, $randomPhoneNumberExt, $randomCity, $randomStreetName, $randomStreetAddress, $randomCountry, $randomCountryCode, $randomLatitude, $randomLongitude

Negocio: $randomCompanyName, $randomCompanySuffix, $randomBs, $randomBsAdjective, $randomBsBuzz, $randomBsNoun

Financieras: $randomBankAccount, $randomBankAccountName, $randomCreditCardMask, $randomBankAccountBic, $randomBankAccountIban, $randomTransactionType, $randomCurrencyCode, $randomCurrencyName, $randomCurrencySymbol, $randomBitcoin

Ventas: $randomPrice, $randomProduct, $randomProductAdjective, $randomProductMaterial, $randomProductName, $randomDepartment

Frases con gancho: $randomCatchPhrase, $randomCatchPhraseAdjective, $randomCatchPhraseDescriptor, $randomCatchPhraseNoun

Gramática: $randomNoun, $randomVerb, $randomIngverb, $randomAdjective, $randomWord, $randomWords, $randomPhrase

Lorem Ipsum: $randomLoremWord, $randomLoremWords, $randomLoremSentence, $randomLoremSentences, $randomLoremParagraph, $randomLoremParagraphs, $randomLoremText, $randomLoremSlug, $randomLoremLines

Solicitudes

- Una solicitud al API permite ponerse en contacto con un servidor con puntos finales del API y realizar alguna acción. La acciones son métodos HTTP.
- Los métodos más comunes son GET, POST, PUT y DELETE. Los nombres de los métodos se explican por sí mismos. Por ejemplo, GET le permite recuperar datos de un servidor. POST le permite agregar datos a un archivo o recurso existente en un servidor. PUT le permite reemplazar un archivo o recurso existente en un servidor. Y DELETE le permite eliminar datos de un servidor.
- Postman simplifica el envío de solicitudes de API. En lugar de probar las API a través de una línea de comando o terminal, ofrece una interfaz gráfica intuitiva que es rápida de aprender y facil de dominar.

Peticiones

- Método: indica la acción HTTP que se desea efectuar sobre el recurso identificado
- URL: identificador del recurso, opcionalmente puede contar con Path Variables (/:id/) y Query Params (?clave1=valor1& clave2=valor2).
- Params: pares clave-valor, agrupados en Path Variables y Query Params, Postman combina todo en la cadena de consulta anterior (URL).
- Headers: pares clave-valor, HTTP cuenta con un amplio conjunto de cabeceras y valores predefinidos, se pueden crear cabeceras personalizadas. Postman proporciona sugerencias de encabezados HTTP comunes y sus valores. Manage Presets permite cachear cabeceras predefinidas.
- Cookies: Puede administrar cookies en aplicaciones nativas utilizando el administrador de cookies para editar las cookies asociadas con cada dominio.
  - nombre=valor; path=/; domain=localhost; HttpOnly; Expires=Tue, 19 Jan 2038 03:14:07 GMT;

Peticiones

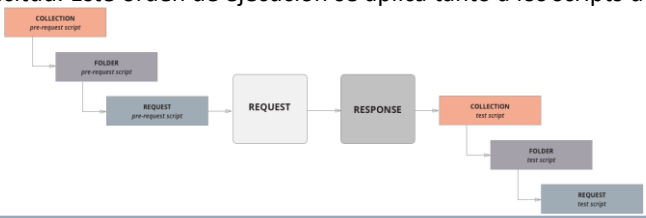
- Authorization: Permite establecer la autenticación, las opciones son: No Auth, Inherit auth from parent, Basic auth, Digest Auth, NTLM Authentication, Bearer Token, OAuth 1.0, OAuth 2.0, Hawk Authentication, AWS Signature.
- Body (Request): Cuerpo de la petición, acepta la opciones: none (sin cuerpo), form-data (simula el relleno de un formulario codificado en multipart/form-data por lo que acpta el envio de ficheros), x-www-form-urlencoded (pares clave-valor similares a los query params), raw (formato libre pero debe establecerse el Content-Type: text, JSON, XML, HTML, Javascript), binary (hay que seleccionar un fichero), GraphQL (cuerpo en formato de consulta GraphQL).
- Pre-request scripts: Fragmentos de código asociados con una solicitud de recopilación que se ejecutan antes de enviar la solicitud.
- Test: Fragmentos de código que se ejecutan después de realizar la petición y recibir la respuesta, contienen las aserciones asociadas a la prueba.

Respuestas

- El visor de respuestas de Postman ayuda a garantizar la exactitud de las respuestas API.
- Una respuesta API consiste en el cuerpo, las cookies, los encabezados y el código de estado.
- Postman organiza el cuerpo, las cookies y los encabezados en diferentes pestañas. El código de estado, el tiempo de duración de la llamada API y el tamaño de la respuesta están visibles junto a las pestañas.
- La pestaña test contiene los resultados de los script de test que se ejecutaron contra la solicitud.
- Cuando se trabaja con colecciones, la respuesta se puede guardar como un ejemplo a efectos de documentación.

Secuencia de comandos

- Postman contiene un poderoso motor de ejecución basado en Node.js que permite agregar un comportamiento dinámico a las solicitudes y colecciones. Esto permite escribir conjuntos de pruebas, crear solicitudes que pueden contener parámetros dinámicos, pasar datos entre solicitudes y mucho más.
- Puede agregar código JavaScript para ejecutar durante dos puntos del flujo:
  - Antes de que una solicitud se envíe al servidor, como un script de pre-solicitud (pestaña Pre-request Script).
  - Después de recibir una respuesta, como un script de prueba (pestaña Tests).
- En caso de que estén disponibles, para cada solicitud de una colección, los scripts siempre se ejecutarán de acuerdo con la siguiente jerarquía: script de nivel de colección, script de nivel de carpeta, script de nivel de solicitud. Este orden de ejecución se aplica tanto a los scripts de solicitud previa como de prueba.



Objecto pm

- El objeto pm contiene toda la información relacionada con el script que se está ejecutando y permite a uno acceder a una copia de la solicitud que se envía o de la respuesta recibida.
- También permite obtener y establecer el entorno y las variables globales.
- Dispone de métodos para enviar solicitudes, crear casos de pruebas y aserciones.

Propiedades de pm

- pm.info: contiene información perteneciente al script que se está ejecutando como el nombre de la solicitud, el ID de la solicitud y el recuento de iteraciones se almacenan dentro de este objeto.
- pm.variables: da acceso a todas las variables se ajustan a una jerarquía específica y permite modificar sus valores en función de su ámbito.
- pm.environment: da acceso a las variables de entorno y permite modificar sus valores.
- pm.globals: da acceso a las variables de globales y permite modificar sus valores.
- pm.request: es una representación de la solicitud (URL y cabeceras) para la que se ejecuta el script. Para un script de pre-solicitud es la solicitud que está a punto de enviarse y en un script de prueba es la representación de la solicitud que se envió.
- SOLO están disponibles en los scripts de prueba:
  - pm.iterationData: da acceso a los valores actuales de variables obtenidos del archivo de datos proporcionado durante una ejecución de recopilación.
  - pm.cookies: da acceso a los valores almacenados en las cookies.

pm.response

- Solo disponible dentro de los scripts de prueba, contiene toda la información relacionada con la respuesta que se recibió.
- Los detalles accesibles de la respuesta son:

  pm.response.code → Number: Versión numérica del HTTP Status Code.

  pm.response.reason() → String: Versión textual del HTTP Status Code.

  pm.response.headers → HeaderList: Colección con las cabeceras recibidas (Objetos: {key:"", value:""}).

  pm.response.responseTime → Number: Tiempo de duración de la llamada API, en milisegundos.

  pm.response.text() → String: Versión de textual del cuerpo de la respuesta para su tratamiento como cadena.

  pm.response.json() → Object: Objeto JavaScript obtenido al hacer la des serialización con JSON.parse() del cuerpo de la respuesta (el content-type debe ser JSON) para su tratamiento como objeto.

Scripts previos a la solicitud

- Los scripts de pre-solicitud son fragmentos de código JavaScript asociados con una solicitud de recopilación que se ejecutan antes de enviar la solicitud.
- Esto es perfecto para los casos de uso que requieran una preparación previa a la solicitud, como incluir una marca de tiempo en los encabezados de solicitud, dar formato a los datos de salida o incluir una traza en el log.

```
console.info('Inicio de la prueba’);
pm.variables.set('ahora', new Date());
```

- Se puede agregar scripts de pre-solicitud a una colección, una carpeta o una sola solicitud. Los script de colección y carpeta permiten reutilizar el código ejecutado comúnmente antes de cada solicitud.

Scripts de prueba

- Los scripts de prueba son fragmentos de código JavaScript asociados con una solicitud de recopilación que se ejecutan después de recibir la respuesta de la solicitud. Aunque están destinados a implementar los casos de prueba pueden incluir trazas en el log y la captura y cacheo en variables de los valores recibidos.
- Para añadir las especificaciones de prueba se utiliza el método pm.test. El método recibe una cadena con el título o descripción de la especificación y la función que contiene el caso de prueba y asegura que el resto del script no se bloqueara, incluso si hay errores dentro de la función.

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

- La función con el caso de prueba puede recibir como parámetro la función con la que indicar que una prueba asíncrona a concluido:

```
pm.test('async test', function (done) {
    setTimeout(() => { pm.expect(pm.response.code).to.equal(200); done(); }, 1500);
 });
```

- Con pm.test.index() se obtiene el número total de pruebas de una ubicación específica.

Scripts de prueba

- Una especificación contiene una o más expectativas (algo que se espera) que ponen a prueba el estado de la respuesta. Una expectativa es una afirmación que debe ser verdadera pero puede ser falsa.
- Una especificación con todas las expectativas verdaderas es una especificación que pasa la prueba, pero con una o más falsas es una especificación que falla.
- Las expectativas se construyen con el método pm.expect que obtiene un valor real de una expresión y se encadenan con afirmaciones del lenguaje natural para compararlo con el valor esperado (constante).

  pm.expect(valor actual).to.equal(valor esperado);
- Postman utiliza la biblioteca ChaiJS (https://www.chaijs.com/) con el estilo apto para cadenas del BDD que proporciona un lenguaje expresivo y un estilo legible.

Expectativas

- Los siguientes elementos se proporcionan como captadores encadenables para mejorar la legibilidad de sus afirmaciones:

  .to .be .been .is .that .which .and .has .have .with .at .of .same .but .does .still

.not: Niega todas las afirmaciones que siguen en la cadena.

.equal(val): verifica si ambos objetos son idénticos ===.

.eql(obj): verifica si ambos objetos son idénticos === (búsqueda profunda).

.ok: verifica si el valor se evalúa como verdadero.

.true: verifica si el valor es verdadero.

.false: verifica si el valor es falso.

.null: verifica si el valor es nulo.

.undefined: verifica si el valor es indefinido.

.NaN: verifica si el valor es NaN.

Expectativas

.exist: verifica si el valor es distinto de null y undefined.
.empty: verifica si el valor de la propiedad length es 0.
.finite: verifica si el valor es distinto de NaN y NaN.
.arguments: verifica si el valor es la propiedad arguments de una función.
.above(n): verifica si el valor actual es mayor que el esperado.
.least(n): verifica si el valor actual es mayor o igual que el esperado.
.below(n): verifica si el valor actual es menor que el esperado.
.most(n): verifica si el valor actual es menor o igual que el esperado.
.within(start, finish): verifica si el valor actual esta en el rango esperado.
.oneOf(list): verifica si el valor actual es uno de la lista esperada.
.a(type): verifica si el valor actual es del tipo esperado.

Expectativas

.instanceof(constructor): verifica si el objeto actual es una instancia del tipo esperado.
.include(val): verifica si el objeto actual incluye el esperado.
.string(str): verifica si la cadena actual contiene el esperado.
.property(name[, val]): verifica si el objeto actual es tiene la propiedad esperado o el valor de la propiedad es el esperado.
.keys(key1[, key2[, …]]): verifica si el objeto actual contiene las propiedades o metodos esperados.
.lengthOf(n): verifica si la longitud del valor actual es mayor es la esperada.
.match(regex): verifica si el valor pertenece al patrón establecido.
.throw([errorLike], [errMsgMatcher]): verifica si una función lanza una excepción.

Expectativas personalizadas

.satisfy (matcher]): el método recibe la función matcher que con el valor actual debe devolver verdadero si satisfice la aserción o falso si debe fallar.

```
expect(value).to.satisfy(function(num) {
  return num > 0;
});
```

.fail([message]): Marca directamente la expectativa como fallida y por lo tanto también la especificación. Se utiliza en las verificaciones por código que desean utilizar el mecanismo de fallos de las expectativas:

```
if …
      expect.fail();
```


API de respuesta

- pm.response.to.have.status(code:Number)
- pm.response.to.have.status(reason:String)
- pm.response.to.have.header(key:String)
- pm.response.to.have.header(key:String, optionalValue:String)
- pm.response.to.have.body()
- pm.response.to.have.body(optionalValue:String)
- pm.response.to.have.body(optionalValue:RegExp)
- pm.response.to.have.jsonBody()
- pm.response.to.have.jsonBody(optionalExpectEqual:Object)
- pm.response.to.have.jsonBody(optionalExpectPath:String)
- pm.response.to.have.jsonBody(optionalExpectPath:String, optionalValue:*)
- pm.response.to.have.jsonSchema(schema:Object)
- pm.response.to.have.jsonSchema(schema:Object, ajvOptions:Object)

API de respuesta

- pm.response.to.be.info: Comprueba el código de estado 1XX
- pm.response.to.be.success: Comprueba el código de estado 2XX
- pm.response.to.be.redirection: Comprueba el código de estado 3XX
- pm.response.to.be.clientError: Comprueba el código de estado 4XX
- pm.response.to.be.serverError: Comprueba el código de estado 5XX
- pm.response.to.be.error: Comprueba el código de estado 4XX o 5XX
- pm.response.to.be.ok: El código de estado debe ser 200
- pm.response.to.be.accepted: El código de estado debe ser 202
- pm.response.to.be.badRequest: El código de estado debe ser 400
- pm.response.to.be.unauthorized: El código de estado debe ser 401
- pm.response.to.be.forbidden: El código de estado debe ser 403
- pm.response.to.be.notFound: El código de estado debe ser 404
- pm.response.to.be.rateLimited: El código de estado debe ser 429

pm.sendRequest

- El método pm.sendRequest permite enviar solicitudes HTTP/HTTPS de forma asíncrona. Con scripts asíncronos, se puede ejecutar la lógica en segundo plano si una tarea es computacional pesada o se están enviando múltiples solicitudes. En lugar de esperar a que se complete una llamada y bloquear las siguientes solicitudes, se puede designar una función de devolución de llamada y recibir una notificación cuando haya finalizado una operación subyacente.
- El método acepta una solicitud compatible con el SDK de recopilación y una devolución de llamada. La devolución de llamada recibe dos argumentos, un error (si corresponde) y una respuesta compatible con SDK.
- Se puede utilizar en el script de pre-solicitud o en el script de prueba.

```
pm.sendRequest('https://postman-echo.com/get', function (err, res) {
  if (err) { console.log(err); }
  pm.test('response should be okay to process', function () {
    pm.expect(err).to.equal(null);
    pm.expect(res).to.have.property('code', 200);
    pm.expect(res).to.have.property('status', 'OK');
  });
});
```

Ejemplos

```
pm.test("response is ok", function () {
   pm.response.to.have.status(200);
});

pm.test("environment to be production", function () {
   pm.expect(pm.environment.get("env")).to.equal("production");
});

pm.test("response should be okay to process", function () {
   pm.response.to.not.be.error;
   pm.response.to.have.jsonBody("");
   pm.response.to.not.have.jsonBody("error");
});

pm.test("response must be valid and have a JSON body", function () {
    pm.response.to.be.success; // info, success, redirection, clientError,  serverError, are other variants
    pm.response.to.be.withBody;
    pm.response.to.be.json;
});
```


Fragmentos

- Si bien hay muy pocas cosas que recordar al escribir pruebas, Postman intenta facilitar el proceso al enumerar fragmentos de uso común al lado del editor.
- Se puede seleccionar el fragmento que se desea agregar y el código apropiado se completa en el editor de prueba.
- Esta es una excelente manera de crear rápidamente casos de prueba.
- Se dispone de SNIPPETS para recuperar y modificar variables de diferentes ámbitos, casos de prueba que evalúan de diferentes formas el cuerpo, las cabeceras, el estado, la duración, …

Ver resultados

- Postman ejecuta pruebas cada vez que ejecuta una solicitud.
- Los resultados se muestran en una pestaña Test del visor de respuestas.
- El encabezado de la pestaña muestra cuántas pruebas pasaron, y los resultados de las pruebas se enumeran aquí.
- Si la prueba se evalúa como verdadera, la prueba pasó.
- Se puede filtrar los resultados por pruebas pasadas o fallidas.

Ejecución de colección

- Las colecciones son grupos de solicitudes que se pueden ejecutar juntas como una serie de solicitudes, en un entorno correspondiente.
- Ejecutar una colección es útil cuando se desea automatizar las pruebas de API. Cuando se ejecuta una colección, envían todas las solicitudes de la colección una tras otra.
- Cuando se usan scripts, se pueden crear conjuntos de pruebas de integración, pasar datos entre solicitudes de API y crear flujos de trabajo que reflejen un caso de uso real de API.
- Para ejecutar una colección se utiliza la utilidad Collection Runner donde habrá que indicar:
  - La colección o carpeta que desea ejecutar.
  - El entorno a utilizar cuando se ejecuta una colección.
  - El número de veces que se ejecutará la colección.
  - El intervalo (en milisegundos) entre cada solicitud en una ejecución de recopilación.
  - El nivel de registro de respuestas cuando se ejecuta la colección.
  - Proporciona un archivo de datos para usar en la ejecución de la recopilación.
  - Si los valores de las variables son persistentes (pruebas irrepetibles)
  - Tratamiento de las cookies.

Archivos de datos

- El uso de archivos de datos es una potente forma de probar el comportamiento de las API en múltiples escenarios.
- Un archivo de datos contiene los valores de sustitución de las variables para cada iteración de una ejecución de colección.
- Actualmente Postman admite archivos JSON y CSV.
- En la forma típica de CSV, la primera fila representa todos los nombres de variables, y las filas siguientes representan valores para estas variables para cada iteración.

```
var1,var2,rslt
11,12,1
21,22,2
31,32,3
```

- En formato JSON, el fichero contendrá un array de objetos donde el nombre de la propiedad será el nombre de la variable y el valor de la propiedad será el valor de la variable:

```
[{ "var1": 11, "var2": 12, "rslt": 1 }, { "var1": 21, "var2": 22, "rslt": 2 }]
```

- Solo se puede usar un archivo de datos para una ejecución y es común para todas las solicitudes por lo que debe contar con todos los valores necesarios.

Flujos de trabajo

- Cuando se inicia una ejecución de recopilación, todas las solicitudes se ejecutan en el orden en que se ven en la aplicación principal, por defecto se ejecutan primero por orden las carpetas y luego cualquier solicitud en la raíz de la colección.
- Se puede alterar el orden de la colección arrastrando y soltando en el panel de navegación de la aplicación principal.
- Para una ejecución concreta se puede alterar el orden en el Collection Runner arrastrando y soltando en el panel RUN ORDER.
- Sin embargo, se puede anular este comportamiento utilizando la función integrada postman.setNextRequest(). Esta función permite especificar como argumento el nombre o ID de la solicitud que se desea ejecutar a continuación y establecer una lógica condicional u omitir solicitudes innecesarias.

```
postman.setNextRequest("request_name");
```

Flujos de trabajo

- Solo funciona con Collection Runner y Newman donde la intención es ejecutar una colección, en lugar de enviar una sola solicitud.
- Siempre se ejecuta al final de la solicitud actual, por lo que se puede colocar en cualquier punto de los scripts de pre-solicitud o test, sin interferir con el resto del código. Si hay más de una asignación, el último valor establecido tiene prioridad.
- Si se ejecuta una colección, se puede saltar a cualquier solicitud de la colección (incluso solicitudes dentro de carpetas). Sin embargo, si se ejecuta una carpeta, el alcance se limita a dicha carpeta y sus subcarpetas.
- Si una solicitud no indica la siguiente se pasa por defecto a la ejecución lineal y continúa por orden con la siguiente solicitud.
- Para detener la ejecución del flujo de trabajo:

  postman.setNextRequest(null);

Integración de línea de comando

- Newman es una utilidad de línea de comandos para ejecutar colecciones de Postman y permite ejecutar y probar una Colección Postman directamente desde la línea de comandos. Está destinado a permitir integrar las pruebas de Postman con los servidores de integración continua y sistemas de compilación (Jenkins, Travis CI, AppVeyor, …).
- Newman se mantiene sincronizado con las características de Postman para permitir ejecutar colecciones de la misma forma en que se ejecutan dentro del Collection Runner de la aplicación Postman.
- Newman se basa en Node.js por lo que es necesario tenerlo instalado. Para instalar Newman:

  npm install -g newman
- Para ejecutar una colección es necesario exportarla a su fichero JSON. Una vez exportada se ejecuta con el comando:

  newman run mycollection.json
- Newman ofrece un amplio conjunto de opciones para personalizar la ejecución:

  newman run -h

Documentación

- La función de documentación de API de Postman permite ver la documentación privada del API o compartir documentación pública del API en una página web.
- Postman genera y aloja automáticamente en tiempo real la documentación basada en navegador del API para las colecciones. Cada colección tiene una vista de documentación pública y privada que Postman genera a partir de datos sincronizados en los servidores (es necesario tener un usuario registrado).
- La documentación para el API incluye:
  - Solicitudes de muestra, encabezados y otros metadatos
  - Descripciones asociadas con solicitudes, carpetas y colecciones.
  - Fragmentos de código generados en algunos de los lenguajes de programación más populares.
- Postman utiliza la ordenación de las solicitudes y carpetas para organizar la documentación en secciones y reflejar la estructura de la colección.
- Se pueden personalizar las descripciones utilizando el estilo Markdown con gráficos incrustados para complementar su documentación.

Monitorización

- La supervisión de Postman permite ejecutar una colección periódicamente para verificar su rendimiento y respuesta, proceso de prueba continuo. Se puede configurar un monitor para que se ejecute con una frecuencia de 5 minutos para verificar si todas las solicitudes de la colección están activas y en buen estado.
- Cuando se configura un monitor, los servidores de Postman realizaran las solicitudes de la colección de acuerdo con la frecuencia especificada. También se puede seleccionar un entorno correspondiente para usar y almacenar variables. Si hay pruebas escritas para las solicitudes, el monitor ejecutará estas pruebas para validar la respuesta y notificar cuando falle una prueba. También se puede configurar cómo recibir las alertas de una gran cantidad de integraciones disponibles.
- Cada usuario de Postman recibe 1.000 llamadas de monitoreo gratis al mes. Las cuentas de pago tienen límites más altos: los equipos Postman Pro tienen 10.000 llamadas de monitoreo mensuales incluidas y los equipos Enterprise tienen 100.000 solicitudes mensuales incluidas.

Servidores simulados

- Los retrasos en el front-end o back-end dificultan que los equipos dependientes completen su trabajo de manera eficiente. Los servidores simulados de Postman pueden aliviar esos retrasos en el proceso de desarrollo.
- Antes de enviar una solicitud real, los desarrolladores front-end pueden crear un mock server para simular cada punto final y su respuesta correspondiente en una Colección Postman. Los desarrolladores pueden ver las posibles respuestas, sin tener que acceder al back-end.
- Postman le permite crear dos tipos de servidores simulados: privados (hay que pasar una clave API en el encabezado x-api-key de la solicitud) y públicos.
  - https://{{mockId}}.mock.pstmn.io/{{mockPath}}
- Los servidores simulados residen en la infraestructura de Postman por lo que es necesario tener un usuario registrado (con limites en el número de solicitudes).
- Para crear un mock server se puede tomar como partida los ejemplos de una colección existente o crear manualmente las solicitudes y respuestas (genera una colección asociada al mock server).
- Una vez creados los mock server no se pueden modificar, pero se pueden crear tantos como diferentes escenarios sean necesarios y asociarlos a la misma colección.

https://www.soapui.org

SOAPUI

Contenidos

- Instalación
- Estructura de proyectos
- Preparación de la prueba
- Service Mocking
- Pruebas funcionales: Pasos, Aserciones, Propiedades
- Ejecución
- Pruebas de carga
- Informes
- Data Driven Testing
- TestRunner Command-Line

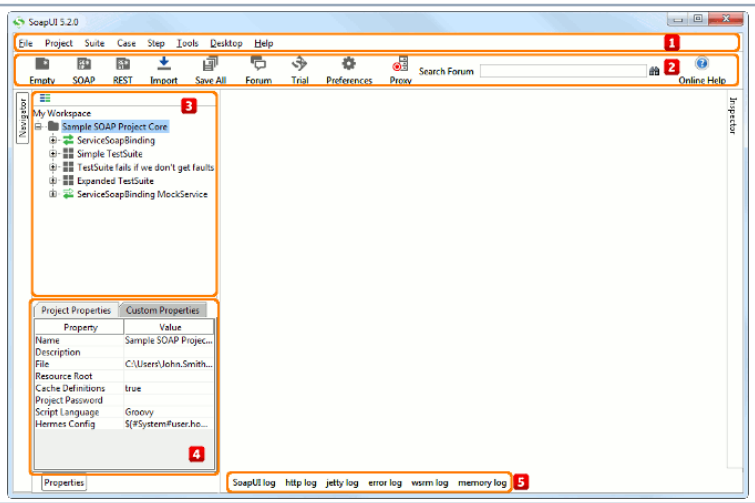
Introducción

- SoapUI es una herramienta para probar servicios web que pueden ser servicios web SOAP, servicios web RESTful u otros servicios basados en HTTP.
- SoapUI es una herramienta de código abierto y completamente gratuita con una versión comercial, SoapUI Pro, que tiene una funcionalidad adicional para empresas con servicios web de misión crítica.
- SoapUI se considera el estándar de facto para las Pruebas de servicio API. Esto significa que hay mucho conocimiento en la red sobre la herramienta y blogs para obtener más información sobre el uso de SoapUI en la vida real.
- SoapUI permite realizar pruebas funcionales, pruebas de rendimiento, pruebas de interoperabilidad, pruebas de regresión y mucho más. Su objetivo es que la prueba sea bastante fácil de comenzar, por ejemplo, para crear una Prueba de carga, simplemente hay que hacer clic derecho en una prueba funcional y ejecutarla como una prueba de carga.
- SoapUI puede simular servicios web (mocking). Se puede grabar pruebas y usarlas más tarde.
- SoapUI puede crear apéndices de código desde el WSDL. Incluso puede crear especificaciones REST (WADL) a partir de la comunicación grabada.

Instalación

- SoapUI está basado en Java, por lo que se ejecuta en la mayoría de los sistemas operativos. Ha sido probado en varias versiones de Windows, así como en Mac y múltiples dialectos de Linux.
- SoapUI requiere una versión 1.6+ de JRE (Java Runtime Environment), se recomienda al menos 1 GB de memoria y aproximadamente 100 MB de espacio en disco.
- Para descargar e instalar accedemos a su página oficial:
  - https://www.soapui.org/downloads/soapui.html
- Si se está instalando con el instalador o las distribuciones independientes, el JRE está incluido y no es necesario en el sistema. De lo contrario, hay que asegúrese de que esté instalado y la variable de entorno JAVA_HOME esté configurada.

Interfaz de SoapUI



Log

- Las diferentes pestañas de log disponibles con las siguientes:
  - SoapUI log: Notificaciones generales y mensajes.
  - Http log: Muestra los datos enviados y recibidos por http. Deshabilitado durante las pruebas de stress.
  - Jetty log: Relacionado con las notificaciones de estado del mock-service.
  - Script log: Los scripts lanzan estos mensajes usando el objeto log disponible (está deshabilitado durante las pruebas de stress pero puede ser habilitado desde /File/Preferencess/UI settings).
  - Error log: Es un log con información sobre los errores ocurridos durante la ejecución. No tienen porque ser solo errores de soapUI, sino que pueden ser producidos por algún servicio o servidor que no esté disponible.
  - Memory Log: Muestra información del uso de la memoria.
- SoapUI utiliza log4j para crear los logs, es posible adaptar la configuración de log4j, renombrando el archivo log4j.xml, llamándolo “soapui-log4j.xml” y posteriormente moviéndolo al directorio bin de soapUI.

Espacio de trabajo

- En SoapUI, el trabajo se organiza en espacios de trabajo y proyectos. Un proyecto puede contener cualquier número de pruebas funcionales, pruebas de carga y simulaciones de servicio requeridas para los propósitos de prueba.
- SoapUI ofrece dos formatos de proyecto:
  - Proyectos independientes (predeterminado): los almacena como un único archivo XML que contiene todos los artefactos del proyecto, como interfaces, pruebas, servicios simulados, scripts, etc.
  - Proyectos compuestos (SoapUI Pro): para equipos de prueba, esto permite que varias personas trabajen en el proyecto al mismo tiempo.

Proyectos

- Los proyectos SOAP se pueden crear desde un archivo WSDL o una llamada de servicio individual. Se puede usar estos proyectos para probar cada aspecto de sus servicios SOAP, verificar que los servicios sean compatibles con los estándares de uso común como WS-Security, WS-Addressing y MTOM, crear pruebas funcionales y de carga, y mucho más.
- Los proyectos REST se pueden crear desde un archivo WADL o, directamente, URI y sus parámetros. Se puede usar estos proyectos para probar servicios RESTful, crear varias solicitudes y verificar la información que recibe, probar una multitud de métodos y operaciones, etc.
- Se pueden importar proyectos existentes locales, compuestos, comprimidos o remotos. Al importar, SoapUI lo comprueba para verificar que sea coherente y que tenga todas las dependencias externas necesarias disponibles. Este proceso se llama resolución .

Propiedades del proyecto

- La propiedad de raíz de recursos controla cómo SoapUI maneja las rutas para los recursos del proyecto.
  - Un camino absoluto: Usa este camino absoluto.
  - $ {projectDir}: Resuelve archivos relativos a la carpeta del proyecto.
  - $ {workspaceDir}: Resuelve archivos relativos a la carpeta que contiene el archivo del espacio de trabajo
- Se pueden crear propiedades personalizadas.
- Se puede usar la propiedad del proyecto Contraseña del proyecto para cifrar todo el contenido del archivo del proyecto (el icono del proyecto contendrá un pequeño carácter E para indicar que se ha cifrado). Al abrir el proyecto SoapUI pedirá dicha contraseña. Para eliminar el cifrado, hay que borrar el valor de la contraseña del proyecto y guardar el proyecto. SoapUI no proporciona ningún medio para recuperar un archivo de proyecto cifrado si se ha perdido la contraseña.

Estructura del proyecto

- Interfaces
  - SOAP
  - REST
- Suite de pruebas (TestSuite)
  - Casos de prueba
    - Pasos de la prueba (pruebas funcionales)
    - Pruebas de carga
    - Pruebas de seguridad
- Simulaciones de servicio
  - SOAP MockServer
  - REST MockServer

Preparación de la prueba

- Importar definición y configurar endpoints:
  - Añadir WSDL (http://www.dneonline.com/calculator.asmx?wsdl)
  - Añadir WADL
  - Importar Swagger (https://petstore.swagger.io/v2/swagger.json)
- Explorar el servicio:
  - Cada servicio basado en WSDL expone una serie de operaciones que tienen un formato de mensaje de solicitud y respuesta (ambos opcionales).
  - Cada servicio REST expone una serie de URLs y el método de solicitud determina la operación a realizar. Las básicas son: POST, GET, PUT y DELETE; aunque SoapUI también admite solicitudes HEAD, OPTIONS, TRACE, PATCH, PROPFIND, LOCK, UNLOCK, COPY y PURGE.
- Preparar y probar peticiones al servicio
  - Para invocar una operación, puede agregar cualquier número de objetos de solicitud a una operación en el árbol del navegador.

SOAP Service Mocking

- La funcionalidad de SOAP Service Mocking en SoapUI le permite crear una simulación compatible con los estándares de un servicio basado en WSDL solo a partir de su contrato WSDL, que en SoapUI se denomina "MockService". Se puede ejecutar directamente desde dentro de SoapUI, con la utilidad de línea de comandos o con un contenedor de servlet estándar.
- Los escenarios de uso son muchos:
  - Prototipos rápidos de servicios web: generar una implementación simulada completa estática de un WSDL en segundos y agregar funcionalidad dinámica usando Groovy. Esto permite implementar y probar clientes mucho más rápido que si se hubiera tenido que esperar a que se desarrollara la solución real.
  - Prueba o desarrollo del cliente: crear implementaciones simuladas de las operaciones deseadas y configurar una serie de respuestas alternativas (incluidos scripts, archivos adjuntos y encabezados http personalizados). Los clientes pueden desarrollarse contra el MockService y probarse sin acceso a los servicios en vivo. Las respuestas pueden ciclarse, aleatorizarse o seleccionarse con la expresión XPath de la solicitud entrante
  - Desarrollo dirigido por pruebas: crear pruebas funcionales y de carga en SoapUI contra un MockService antes o durante la implementacíon los servicios reales
- Un MockService cumple con los estándares aceptados WSDL, SOAP o HTTP y un cliente debe poder usarlo como si fuera un servicio real.

SOAP Service Mocking

- Un MockService puede simular cualquier número de contratos WSDL y la funcionalidad de scripting incorporada le permite simular básicamente cualquier tipo de comportamiento deseado; respuestas fijas, errores aleatorios, resultados dinámicos, etc. Incluso es posible implementar un servicio completo en SoapUI y desplegarlo en un contenedor de servlet estándar utilizando la funcionalidad DeployAsWar (aunque no se recomienda para uso en producción).
- MockServices proporciona la simulación del servicio al exponer un número arbitrario de MockOperations que, a su vez, pueden contener un MockRequest Dispatcher y cualquier número de mensajes MockResponse.
- Cuando el MockService recibe una solicitud SOAP y se envía a una MockOperation específica y se selecciona la MockResponse correspondiente en función del MockRequest Dispatcher configurado.
- Cada MockOperation se corresponde a una Operación WSDL de un Servicio WSDL del proyecto de prueba, y los mensajes MockResponse contenidos se crean a partir del esquema asociado con la operación. Esto no significa que un MockResponse deba cumplir con el esquema asociado, puede devolver cualquier mensaje de texto o XML arbitrario y, por ejemplo, se puede configurar para que devuelva un mensaje de respuesta para una operación completamente diferente, lo que permite probar en los clientes la 'capacidad para manejar mensajes de respuesta no válidos e inesperados.

SOAP Service Mocking

- El MockResponse es el mensaje que MockService debe devolver al cliente que realiza la llamada. MockResponse puede contener contenido personalizado, encabezados y archivos adjuntos, lo que le permite simular cualquier tipo de respuesta HTTP válida (o no válida), y las posibilidades de secuencias de comandos adicionales permiten agregar fácilmente contenido dinámico a la respuesta saliente. Los MockResponse se pueden generar desde los servicios reales.
- Una vez que MockService ha recibido una solicitud y se ha enviado a una MockOperation, SoapUI debe seleccionar el método de respuesta correcto y devolverlo al cliente. Hay varios métodos de envío diferentes:
  - SECUENCIA: este es el método de envío más simple; las respuestas se seleccionan y devuelven en el orden en que se agregaron al MockOperation.
  - ALEATORIO: casi tan simple, este despachador selecciona qué respuesta usar al azar, con el tiempo todas las respuestas se habrán devuelto la misma cantidad de veces.
  - QUERY-MATCH: es bastante versátil, puede devolver diferentes respuestas en función del contenido de la solicitud.
  - XPATH: similar a QUERY_MATCH pero no tan potente, solo permite una condición.
  - SCRIPT: la opción más versátil y más difícil de dominar, se crea un script que debería devolver el nombre de MockResponse para usar.

SOAP Service Mocking

- Al hacer clic en el botón Ejecutar , MockService se inicia inmediatamente dentro de SoapUI, se puede consultar jetty log.
- El MockService ahora está listo para manejar las solicitudes SOAP entrantes en la ruta y el puerto configurados. Estas se enviarán a la correspondiente MockOperation en función de su contenido, si no hay disponible ninguna MockOperation coincidente, se devolverá un fallo SOAP estándar.
- El registro de mensajes en la parte inferior muestra todos los mensajes que ha recibido el MockService desde que se inició por última vez y permite consultar, a través del visor de mensajes, la petición recibida así como la respuesta emitida.
- Además, también expone el WSDL de MockServic en la URL estándar de WSDL; por ejemplo, introduciendo http://localhost:8088/mockSampleServiceBinding?WSDL en un navegador se mostrará el WSDL.

REST Service Mocking

- De forma similar a los SOAP Service Mocking, la función de simulación del servicio REST permite simular un servicio REST creando un servicio simulado. Luego se puede ejecutar directamente desde SoapUI o usar la aplicación de línea de comandos mockservicerunner.bat
- Un servicio simulado simula un servicio en vivo al exponer un cierto número de acciones simuladas (método y URL). Las acciones simuladas (Mock Action), a su vez, contienen una serie de respuestas simuladas.
- Las respuestas simuladas permiten establecer el HTTP Status Code, los valores de las cabeceras, el contenido (response body) y su media-type (content-type).
- El MockRequest Dispatcher esta limitado a SECUENCIA y SCRIPT.
- Los REST Service Mocking se ejecutan y supervisan de la misma forma que los SOAP Service Mocking.

Pruebas funcionales

- En SoapUI, las pruebas funcionales se pueden usar para validar requisitos funcionales, tanto para invocar los Servicios Web propios (=”pruebas unitarias”) como para una secuencia de peticiones (= " pruebas de integración"). Además, es posible añadir lógica a las pruebas mediante scripts de Groovy, lo que permite, por ejemplo, interactuar con una base de datos o realizar un flujo de pruebas complejo.
- Las pruebas funcionales, en SoapUI, se pueden usar para una variedad de propósitos:
  - Pruebas Unitarias: valida que cada operación del Servicio Web funciona como se indica
  - Pruebas de compatibilidad: valida que el resultado devuelto por el Servicio Web es compatible con su definición
  - Prueba de procesos: valida que una secuencia de invocaciones de Servicios Web ejecuta un proceso de negocio requerido
  - Pruebas guiadas por datos: valida que cualquiera de los anteriores funciona como requerimiento de datos de entrada procedentes de fuentes externas (por ejemplo, una base de datos u otro servicio Web)

Pruebas funcionales

- SoapUI soporta pruebas funcionales de Servicios Web suministrando un caso de prueba con un número de pasos que pueden ser ejecutados en secuencia. En la actualidad, hay seis tipos de pasos que proporcionan muchas posibilidades de prueba. Los casos de prueba están organizados en un grupo de pruebas y en un mismo proyecto se pueden crear varios grupos de pruebas.
- SoapUI estructura las pruebas funcionales en tres niveles: TestSuites, TestCases y TestSteps.
- Un TestSuite es una colección de TestCases que se pueden usar para agrupar pruebas funcionales en unidades lógicas. Se puede crear cualquier número de TestSuites dentro de un proyecto de SoapUI para admitir escenarios de pruebas masivas.
- Un TestCase es una colección de TestSteps que se ensamblan para probar algunos aspectos específicos de sus servicios. Puede agregar cualquier número de TestCases a el TestSuite que lo contenga e incluso modularizarlos para llamarse entre sí para escenarios de prueba complejos.
- TestSteps son los "bloques de construcción" de las pruebas funcionales en SoapUI. Se agregan a un TestCase y se utilizan para controlar el flujo de ejecución y validar la funcionalidad de los servicios que se probarán.

Tipos de Paso

- SOAP Request: Envía una petición SOAP y permite que la respuesta sea validada usando una variedad de aserciones
- REST Request: Ejecuta una Request Rest definida en el proyecto
- HTTP Request: Ejecuta una llamada http
- JDBC Request: Ejecuta una petición a una BBDD
- AMF Request: Ejecuta una petición AMF (es el formato de mensajería Adobe ActionScript utilizado por las aplicaciones Flash / Flex para interactuar con un servidor back-end).
- Properties: Se utiliza para definir propiedades globales que pueden ser leídas desde una fuente externa.
- Property Transfer: Utilizadas para transferir los valores de propiedad entre dos TestSteps
- Run TestCase Step: Ejecuta otro TestCase dentro de uno ya existente
- Conditional Goto: Permite cualquier número de saltos condicionales en la ruta de ejecución del TestCase.
- Delay Step: Pausa la ejecución de una TestCase durante un número especificado de milisegundos.
- Groovy Script: Ejecuta un script Groovy que puede hacer más o menos "cualquier cosa"
- Mock Response: Escucha un respuesta SOAP que se valida y devuelve una respuesta mock
- Manual TestStep: Se utiliza cuando se necesita interacción manual del tester dentro de la prueba
- Receive MQTT Message
- Publish using MQTT
- Drop MQTT Connection

TestRequests

- Los TestRequests son una de las principales características cuando se trabaja con SoapUI. Extiende peticiones estándar con la posibilidad de añadir cualquier número de aserciones que se aplicarán a la respuesta recibida por la petición. Esto es, comprueba que la respuesta contenga lo que se espera que contenga.
- Los TestRequests son enviados manualmente a través de las acciones de envío de los editores o cuando se ejecuta el TestCase que contiene la petición.
- La respuesta de la petición es validada contra las aserciones de peticiones y el icono de la petición cambia para reflejar el resultado de la validación; verde significa que todas las validaciones fueron bien y rojo que algunas fallaron. Un icono con el fondo gris indica que la petición todavía no ha sido enviada para validar, un fondo blanco indica que los TestRequests carecen de aserciones.

Petición

- SOAP Request:
  - Vinculada a operación del interfaz contiene el fragmento SOAP de la petición en XML para su edición.
- REST Request:
  - Vinculada a un método del interfaz contiene los parámetros de solicitud HTTP para la asignación de sus valores.
- HTTP Request:
  - Permite crear una petición a una URL usando el método indicado y parámetros de solicitud HTTP para la asignación de sus valores.

Petición

- Todas las peticiones se pueden completar con:
  - Auth : permite establecer el mecanismo de autenticación y la identidad con la que se realizará la petición.
  - Encabezados HTTP: permite agregar los valores a los encabezado de la solicitud
  - Adjuntos: permite agregar ficheros que se enviaran adjuntos con la solicitud
- Las peticiones SOAP adicionalmente se pueden completar con:
  - WS-A : las propiedades utilizadas para agregar encabezados WS-A a Request / MockResponse de acuerdo con la especificación WS Addressing.
  - WS-RM : las propiedades utilizadas para configurar y usar una secuencia WS-RM para la solicitud de acuerdo con la especificación de WS Reliable Messaging.

Respuesta

- Una vez ejecutada manualmente la petición se mostrara la respuesta. Admite diferentes formatos: XML, JSON, HTML, Raw.
- Así mismo permite inspeccionar:
  - Encabezados HTTP : muestra los encabezado HTTP de la respuesta.
  - Adjuntos: muestra los adjuntos de la respuesta asociada.
  - Información SSL: muestra la respuesta detallada SSL para la respuesta actual.
  - WS-A: muestra las propiedades del encabezado WS-Addressing.
  - WSS: muestra las propiedades del encabezado WS-Security

Aserciones

- Las aserciones se utilizan para validar el mensaje recibido por un TestStep durante la ejecución, generalmente comparando partes del mensaje (o el mensaje completo) con algún valor esperado.
- Se puede agregar cualquier cantidad de aserciones a una muestra de TestStep, cada una validando algún aspecto o contenido diferente de la respuesta.
- Después de que se ejecuta una muestra TestStep, todas las aserciones se aplican a la respuesta recibida y si alguna de ellas falla, el TestStep se marca como fallido en la vista del TestCase y se muestra una entrada FALLO correspondiente en el Registro de ejecución de prueba.
- Las aserciones se pueden deshabilitar para que no se apliquen.
- Las aserciones se dividen en varias categorías para facilitar la gestión. Solo se habilitan las categorías que contienen aserciones aplicables para un tipo específico de muestra.

Categoría: contenido de propiedad

- Contains: busca la existencia de un token de cadena en el valor de la propiedad, admite expresiones regulares. Aplicable a cualquier propiedad.
- Not Contains: busca la inexistencia de un token de cadena en el valor de la propiedad, admite expresiones regulares. Aplicable a cualquier propiedad.
- XPath Match: utiliza una expresión XPath para seleccionar contenido de la propiedad de destino y compara el resultado con un valor esperado. Aplicable a cualquier propiedad que contenga XML.
- Xquer Matchy: utiliza una expresión XQuery para seleccionar contenido de la propiedad de destino y compara el resultado con un valor esperado. Aplicable a cualquier propiedad que contenga XML.
- JsonPath Count: usa una expresión JsonPath para contar las ocurrencias de un elemento. Aplicable a cualquier propiedad que contenga JSON.
- JsonPath Existence Match: utiliza una expresión JsonPath para seleccionar contenido de la propiedad de destino y compara el resultado con un valor esperado. Aplicable a cualquier propiedad que contenga JSON.
- JsonPath Match: utiliza una expresión JsonPath para seleccionar contenido de la propiedad de destino y compara el resultado con un valor esperado. Si los valores coinciden con las afirmaciones pasadas, de lo contrario falla. Aplicable a cualquier propiedad que contenga JSON.
- JsonPath RegEx Match: utiliza una expresión JsonPath para seleccionar contenido de la propiedad de destino y compara el resultado con una expresión regular especificada. Aplicable a cualquier propiedad que contenga JSON.

Categoría: Cumplimiento, estado y estándares

- HTTP Download all resource: descarga todos los recursos referidos como un documento HTML (imágenes, scripts, etc.) y valida que estén disponibles. Aplicable a cualquier propiedad que contenga HTML.
- Valid HTTP Status Codes: comprueba que se recibió un resultado HTTP con un código de estado en la lista de códigos definidos. Aplicable a cualquier TestStep que recibe mensajes HTTP.
- Invalid HTTP Status Codes: comprueba que se recibió un resultado HTTP con un código de estado que no figura en la lista de códigos definidos. Aplicable a cualquier TestStep que recibe mensajes HTTP
- SOAP Response: valida que la última respuesta recibida es una respuesta SOAP válida. Aplicable solo a los pasos de solicitud de prueba SOAP.
- SOAP Request: valida que la última solicitud recibida es una solicitud SOAP válida. Aplicable solo a MockResponse TestSteps.
- Not SOAP Fault: valida que el último mensaje recibido no es un fallo SOAP. Aplicable a SOAP TestSteps.
- SOAP Fault: valida que el último mensaje recibido es un fallo SOAP. Aplicable a la solicitud SOAP TestSteps
- Schema Compliance: valida que el último mensaje recibido sea compatible con la definición de esquema WSDL o WADL asociada. Aplicable a SOAP y REST TestSteps.
- WS-Addressing Request : valida que la última solicitud recibida contiene encabezados de direccionamiento WS válidos. Aplicable solo a MockResponse TestSteps.
- WS-Addressing Response: valida que la última respuesta recibida contiene encabezados de WS-Addressing válidos. Aplicable solo a los pasos de solicitud de prueba SOAP.
- WS-Security Status: valida que el último mensaje recibido contenía encabezados WS-Security válidos. Aplicable a SOAP TestSteps.

Otras categorías

- Script
  - Script Assertion: ejecuta una secuencia de comandos personalizada para realizar validaciones arbitrarias. Aplicable solo a TestSteps (es decir, no a las propiedades).
- SLA
  - Response SLA: valida que el último tiempo de respuesta recibido estuvo dentro del límite definido. Aplicable a Script TestSteps y TestSteps que envían solicitudes y reciben respuestas.
- JMS
  - JMS Status: valida que la solicitud JMS del TestStep objetivo se ejecutó correctamente. Aplicable a Request TestSteps con un punto final JMS.
  - JMS Timeout: valida que la instrucción JMS del TestStep de destino no tardó más de la duración especificada. Aplicable a Request TestSteps con un punto final JMS.
- JDBC
  - JDBC Status: valida que la instrucción JDBC del TestStep objetivo se ejecutó correctamente. Aplicable solo a JDBC TestSteps.
  - JDBC Timeout: valida que la instrucción JDBC del TestStep de destino no tardó más de la duración especificada. Aplicable solo a JDBC TestSteps.
- Security
  - Sensitive Information Exposure: comprueba que el último mensaje recibido no expone información confidencial sobre el sistema de destino. Aplicable a REST, SOAP y HTTP TestSteps.

JSONPath

- JSONPath es una forma estandarizada para consultar elementos de un objeto JSON. JSONPath utiliza expresiones de ruta para desplazarse por elementos, elementos anidados y matrices en un documento JSON.
- Cubre solo partes esenciales de XPath 1.0.
- Las expresiones JSONPath siempre se refieren a una estructura JSON de la misma manera que las expresiones XPath se usan en combinación con un documento XML. Dado que una estructura JSON suele ser anónima y no necesariamente tiene un "objeto miembro raíz", JSONPath asume el nombre abstracto $ asignado al objeto de nivel externo.
- Las expresiones JSONPath pueden usar la notación de puntos
  - $.store.book[0].title
- o la notación corchetes
  - $['store']['book'][0]['title']
- JSONPath permite el símbolo comodín * para nombres de miembros e índices de matriz. Toma prestado el operador descendiente '..' de E4X y la sintaxis de corte de matriz [start:end:step] de ECMASCRIPT 4 .
- Las expresiones del lenguaje de script subyacente (<expr>) se pueden usar como una alternativa a los nombres o índices explícitos como en
  - $.store.book[(@.length-1)].title
- usando el símbolo '@' para el objeto actual.
- Las expresiones de filtro son compatibles a través de la sintaxis ?(<boolean expr>) como en
  - $.store.book[?(@.price < 10)].title

JSONPath

- JSONPath es una forma estandarizada para consultar elementos de un objeto JSON. JSONPath utiliza expresiones de ruta similares a XPath para desplazarse por elementos, elementos anidados y matrices en un documento JSON.
- JSONPath es solo cubre las partes esenciales de XPath 1.0.
- Las expresiones JSONPath siempre se refieren a una estructura JSON de la misma manera que las expresiones XPath se usan en combinación con un documento XML. Dado que una estructura JSON suele ser anónima y no necesariamente tiene un "objeto miembro raíz", JSONPath asume el nombre abstracto $ asignado al objeto de nivel externo.
- Las expresiones JSONPath pueden usar la notación de puntos
  - $.store.book[0].title
- o la notación corchetes
  - $['store']['book'][0]['title']

JSONPath

- JSONPath permite el símbolo comodín * para nombres de miembros e índices de matriz.
  - $.store.*
- Toma prestado el operador descendiente '..' de E4X y la sintaxis de recorte de matriz [start:end:step] de ECMASCRIPT 4.
  - $..author
  - $..book[0:]
- Las expresiones del lenguaje de script subyacente (<expr>) se pueden usar como una alternativa a los nombres o índices explícitos, usando el símbolo '@' para el objeto actual, como en
  - $.store.book[(@.length-1)].title
- Las expresiones de filtro son compatibles a través de la sintaxis ?(<boolean expr>) como en
  - $.store.book[?(@.price < 10)].title

JSONPath

XPath	JSONPath	Descripción
/	$	el objeto / elemento raíz
.	@	el objeto / elemento actual
/	. or []	operador hijo
..	n/a	operador padre
//	..	descenso recursivo
*	*	comodín. Todos los objetos / elementos independientemente de sus nombres.
@	n/a	acceso al atributo. Las estructuras JSON no tienen atributos.
[]	[]	operador de subíndice. XPath lo usa para iterar sobre colecciones de elementos y para predicados . En Javascript y JSON es el operador de matriz nativo.
|	[,]	el operador de unión en XPath da como resultado una combinación de conjuntos de nodos. JSONPath permite nombres alternativos o índices de matriz como un conjunto.
n/a	[start:end:step]	operador de recorte de matriz.
[]	?()	aplica una expresión de filtro
n/a	()	expresión de script, utilizando el motor de script subyacente.

JSONPath

XPath	JSONPath	Resultado
/store/book/author	$.store.book[*].author	los autores de todos los libros en la tienda
//author	$..author	todos los autores
/store/*	$.store.*	todas las cosas en la tienda
/store//price	$.store..price	el precio de todo en la tienda
//book[3]	$..book[2]	el tercer libro
//book[last()]	$..book[(@.length-1)] $..book[-1:]	el último libro en orden.
//book[position()< 3]	$..book[0,1] $..book[:2]	los dos primeros libros
//book[isbn]	$..book[?(@.isbn)]	filtrar todos los libros con número isbn
//book[price<10]	$..book[?(@.price<10)]	filtrar todos los libros más baratos que 10
//*	$..*	todos los elementos en el documento XML o todos los miembros de la estructura JSON.

Propiedades

- Las propiedades son un aspecto central de las pruebas más avanzadas con SoapUI.
- En lo que respecta a las propiedades de pruebas funcionales, se utilizan para parametrizar la ejecución y la funcionalidad de sus pruebas, por ejemplo:
  - Las propiedades se pueden utilizar para mantener los puntos finales de sus servicios, lo que facilita el cambio de los puntos finales reales utilizados durante la ejecución de la prueba (consulte el ejemplo a continuación).
  - Las propiedades se pueden usar para mantener las credenciales de autenticación, lo que facilita su administración en un lugar central o en un archivo externo.
  - Las propiedades se pueden usar para transferir y compartir identificadores de sesión durante la ejecución de la prueba, por lo que múltiples pasos de prueba o casos de prueba pueden compartir las mismas sesiones.
- Las propiedades se pueden definir en varios niveles en SoapUI:
  - En el nivel Proyecto, TestSuite y TestCase
  - En un Properties TestStep
  - En un DataGen TestStep
  - Como parte de una configuración de TestStep (DataSource TestStep y DataSink TestStep)
- Además, la mayoría de los otros TestSteps exponen propiedades para lectura y escritura, por ejemplo:
  - Script TestStep expone una propiedad "Result" que contiene el valor de cadena devuelto por el último script ejecutado.
  - Todos los pasos de Request exponen una propiedad con la última respuesta recibida.
- Las propiedades pueden leerse y escribirse fácilmente desde scripts y también transferirse entre TestSteps con Property-Transfer TestStep

Properties TestStep

- El Properties TestStep se utiliza para definir propiedades personalizadas que se utilizarán dentro de un TestCase.
- Sus principales ventajas sobre la definición de propiedades en el nivel TestCase son:
  - Se pueden organizar las propiedades en varios Properties TestStep (si se tienen muchas de ellas).
  - Se puede especificar los nombres de los archivos de origen y destino que se usarán para leer y escribir las propiedades contenidas cuando se ejecute TestStep. En formato .properties:

    Propiedad1=Valor1

    Propiedad2=Valor2

Property Transfer TestStep

- Los Property Transferes son TestStep que transfieren propiedades entre los contenedores de propiedades dentro del mismo alcance que el Property Transfer TestStep (es decir, que contenga su TestCase, su TestSuite, el proyecto y las propiedades del destino).
- El paso puede contener un número arbitrario de "transferencias" especificando una propiedad de origen y destino con expresiones opcionales de XPath/XQuery.
- Property Transferes utiliza el mismo motor de Saxon XPath/XQuery descrito para las aserciones XPath y XQuery.
- Tras la ejecución de un TestCase, cada transferencia se realiza mediante la selección de las propiedades especificadas por el paso de origen, por la propiedad y expresión XPath opcional y copiando su valor a la propiedad especificada por el paso de destino pudiendo usar una expresión Xpath opcional.
- Si las expresiones XPath están especificadas, SoapUI tratará de sustituir el nodo destino con el nodo de origen si son del mismo tipo. Si no (por ejemplo, cuando se asigna texto () a un @atributo), SoapUI hará todo lo posible para copiar el valor.
- El origen de la transferencia puede ser la respuesta de un paso anterior.
- El destino puede ser el contenido de un paso posterior, pero ser mas cómodo el uso de la expansión de propiedades.

Propiedades de Expansión

- SoapUI proporciona una sintaxis común para insertar dinámicamente ("expandir") valores de propiedad durante el procesamiento. La sintaxis es la siguiente:
  - ${[scope]propertyName[#xpath-expression]}
- donde alcance puede ser uno de los siguientes valores literales:
  - #Project#: hace referencia a una propiedad del Proyecto
  - #TestSuite#: hace referencia a una propiedad en el TestSuite que lo contiene
  - #TestCase#: hace referencia a una propiedad en el TestCase que lo contiene
  - #MockService#: hace referencia a una propiedad en el MockService que lo contiene
  - #Global#: hace referencia a una propiedad global en todos los proyectos. En File>Preferences>Global Properties.
  - #System#: hace referencia a una propiedad del sistema. En Help>System properties.
  - #Env#: hace referencia a una variable de entorno
  - [Nombre de TestStep]#: hace referencia a una propiedad TestStep
- Si no se especifica ningún alcance, la propiedad se resuelve de la siguiente manera:
  - Mira en el contexto actual (por ejemplo, TestRunContext) una propiedad con el nombre coincidente
  - Busca una propiedad global coincidente
  - Comprueba si hay una propiedad del sistema coincidente

Propiedades de Expansión

- Si la expansión de la propiedad incluye además una expresión XPath, se usará para seleccionar el valor correspondiente del valor de la propiedad referenciada (que debe contener XML):
  - ${Search Request#Response#//ns1:Item[1]/n1:Author[1]/text()}
- SoapUI 2.5 introdujo la posibilidad de escribir scripts directamente dentro de una PropertyExpansion; si se prefija el contenido con un '=' el contenido restante hasta la llave de cierre se evaluará como un script y se insertará su resultado:
  - ${=(int)(Math.random()*1000)}
- Dependiendo del contexto de la expansión, las variables relevantes estarán disponibles para acceder al modelo de objeto SoapUI.
  - $ {= request.name}
- Las siguientes variables están (casi) siempre disponibles en estos scripts:
  - log: un Logger log4j que registra en la ventana de registro
  - modelItem: el modelItem actual (por ejemplo, Request, MockResponse, etc.).
  - context: el contexto de ejecución actual (por ejemplo, un TestCase o MockService)

JDBC Request

- SoapUI 3.5 presento un nuevo TestStep para recuperar datos de una base de datos utilizando JDBC. El resultado está formateado como XML y se puede afirmar o procesar de la manera estándar (XPath , Xquery, …), pero también hay dos aserciones adicionales disponibles:
  - Afirmación de estado JDBC
  - La aserción JDBC Timeout verificará que el SQL se ejecutó dentro del tiempo de espera configurado.
- Para utilizar JDBC TestStep se deberá agregar el controlador JDBC a la carpeta soapui_home/bin/ext y reiniciar la aplicación.
- Para configurar la conexión:
  - Driver: com.mysql.jdbc.Driver
  - Connection String: jdbc:mysql://localhost:3306/sakila?user=root&password=root
- La consulta SQL, los parámetros pueden ser referenciados mediante la expansión de propiedades:
  - SELECT count(*) FROM `sakila`.`actor` WHERE `actor_id`=:id

Run TestCase TestStep

- Si los escenarios de prueba se vuelven cada vez más complejos, es posible que se desee compartir una serie de TestSteps entre diferentes TestCases, tal vez para configurar algunos requisitos previos (inicio de sesión, etc.) o para realizar ciertos pasos en diferentes contextos.
- Run TestCase TestStep presenta un enfoque flexible para la modularización; ejecuta el TestCase objetivo configurado opcionalmente, pasando y recuperando propiedades antes y después de la ejecución, por ejemplo, se podría usar esto para llamar a una secuencia compleja de TestSteps para realizar y validar un procedimiento de inicio de sesión, pasando las credenciales requeridas y recibiendo el sessionid resultante.

Conditional Goto TestStep

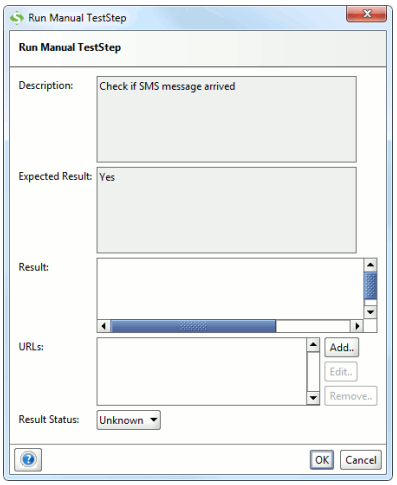
- Los pasos de condicional Goto evalúan un número arbitrario de condiciones XPath del mensaje de respuesta de la petición anterior y transfieren la ejecución del TestCase al TestStep asociado con la primera condición que se evalúe a verdadero.
- Esto permite la ejecución condicional de rutas de TestCase, donde el resultado de algunas peticiones controla cómo moverse por el TestCase.
- Si no hay condiciones que coincidan con la respuesta actual, la ejecución del TestCase continúa después del paso Goto de forma habitual.
- Ejemplo de escenarios:
  - Ramificaciones que dependen de los resultados devueltos por una petición
  - Reiniciar después de un largo retraso en las pruebas de vigilancia
  - En varias ocasiones esperar y chequear el valor de estado antes de continuar (por ejemplo en un proceso por lotes)
- Las condiciones usan el mismo motor que Saxon XPath descrito para las aserciones Xpath (una condición debe devolver un valor booleano para ser válida)

Delay TestStep

- Los TestStep de retardo se pueden insertar en cualquier posición en un TestCase para pausar la ejecución de un TestCase durante un número determinado de milisegundos.

Manual TestStep

- Manual TestStep se utiliza cuando se necesita interacción manual del tester dentro de la prueba. Algunos ejemplos:
  - Iniciar dispositivos como teléfonos movil
  - Realizar una acción en el sitio web o en el servidor
  - Verificar manualmente la base de datos / registros para
- Para definirlo se puede especificar una descripción y el resultado esperado. Cuando la ejecución del caso de prueba alcanza un paso de prueba Manual, se muestra una ventana emergente con la descripción y el resultado esperado y permite agregar los detalles de los resultados reales y especificar el botón de opción Pasar-Fallar-N/A para establecer el estado del paso de prueba. Al hacer clic en continuar, SoapUI va al siguiente paso.
- Estos pasos de prueba manuales, se omiten cuando se inicia a través de testrunner ya que no hay forma de interactuar con ventanas emergentes.



Script Step

- SoapUI ofrece amplias opciones para la creación de secuencias de comandos, utilizando Groovy o Javascript (desde SoapUI 3.0) como su lenguaje de secuencias de comandos, que se utiliza para establecer el nivel del proyecto en la pestaña de detalles del proyecto en la parte inferior izquierda.
- Todos los scripts tienen acceso a una serie de variables específicas de la situación, que siempre incluyen un objeto de registro para iniciar sesión en Groovy Log y un objeto de contexto para realizar PropertyExpansions específicas de contexto o manejo de propiedades, si corresponde.
- Una variable específica de contexto siempre está disponible para acceder directamente al modelo de objetos SoapUI.

Script Step

- Los scripts se pueden usar en los siguientes lugares en SoapUI:
  - Como parte de un TestCase con Script TestStep, que permite que las pruebas realicen prácticamente cualquier funcionalidad deseada
  - Antes y después de ejecutar un TestCase o TestSuite para inicializar y limpiar antes o después de ejecutar sus pruebas.
  - Al iniciar/detener un MockService para inicializar o limpiar el estado de MockService
  - Para proporcionar despacho dinámico de MockOperation o crear contenido dinámico de MockResponse
  - Al abrir / cerrar un proyecto, para inicializar o limpiar configuraciones relacionadas con el proyecto
  - Como un DataSource o DataSink dinámico con los pasos de prueba de DataSource / DataSink correspondientes
  - Para crear aserciones arbitrarias con la aserción de script
  - Para extender SoapUI y agregar funcionalidad arbitraria al núcleo SoapUI.

Ejecución

- Ejecución de prueba
  - Todas las vistas anteriores tienen una barra de herramientas en la parte superior con botones que ejecutan los elementos de prueba contenidos.
- Salida de la Prueba
  - Todas las vistas anteriores también contienen un registro de ejecución en la parte inferior que muestra información continua sobre los pasos de prueba ejecutados y su estado.
- Informes
  - SoapUI Pro también agrega un botón "Crear informe" a la barra de herramientas superior, lo que le permite exportar los resultados de la ejecución actual a un documento, por ejemplo, un PDF.

Pruebas de carga

- Las pruebas de rendimiento son una de las tareas más comunes en las pruebas de servicios web. Lo que se llama una prueba de carga en SoapUI es en realidad una prueba de rendimiento.
- Las pruebas de rendimiento crean o simulan artificialmente la carga y miden cómo la maneja el entorno.
- Esto significa que no necesariamente tiene que ser el rendimiento de un sistema bajo una carga alta, sino también el rendimiento bajo la carga base o la carga esperada.
- En SoapUI, se crean las pruebas de carga sobre la base de pruebas funcionales existentes. Esto ayuda a crear rápida y fácilmente pruebas de rendimiento para el servicio web. Luego se puede validar el rendimiento del servicio web utilizando diferentes estrategias de carga, verificar que su funcionalidad no se rompa bajo carga, ejecutar varias pruebas de carga simultáneamente para ver cómo se afectan entre sí y mucho más.

Pruebas de carga

- Pruebas de línea de base
  - Técnicamente hablando, esto se puede definir como pruebas de rendimiento puro. Las pruebas de línea de base examinan cómo funciona un sistema bajo la carga esperada o normal y crea una línea de base con la que se pueden comparar otros tipos de pruebas.
  - Objetivo: encontrar métricas para el rendimiento del sistema bajo carga normal.
- Prueba de carga
  - La prueba de carga incluye aumentar la carga y ver cómo se comporta el sistema bajo una carga mayor. Durante las pruebas de carga, puede monitorear los tiempos de respuesta, el rendimiento, las condiciones del servidor y más. Sin embargo, el objetivo de las pruebas de carga no es romper el entorno de destino.
  - Objetivo: encontrar métricas para el rendimiento del sistema bajo alta carga.
- Prueba de esfuerzo o estrés
  - La prueba de esfuerzo consiste en aumentar la carga hasta encontrar el volumen de carga donde el sistema realmente se rompe o está cerca de romperse.
  - Objetivo: encontrar el punto de ruptura del sistema.

Pruebas de carga

- Pruebas de remojo
  - Para encontrar inestabilidades del sistema que ocurren con el tiempo, debe ejecutar pruebas durante un período prolongado. Para eso están las pruebas de remojo; ejecute pruebas de carga o incluso pruebas de línea de base durante un largo período de tiempo y vea cómo el entorno de destino maneja los recursos del sistema y si funciona correctamente. El defecto más común encontrado en las pruebas de remojo son las pérdidas de memoria. El escenario más común para las pruebas de remojo es activar una serie de pruebas cuando sale de su oficina un viernes y lo deja correr durante el fin de semana.
  - Objetivo: Asegurarse de que no surja ningún comportamiento no deseado durante un largo período de tiempo.
- Pruebas de escalabilidad
  - El propósito de las pruebas de escalabilidad es verificar si su sistema se adapta adecuadamente a la carga cambiante. Por ejemplo, un mayor número de solicitudes entrantes debería causar un aumento proporcional en el tiempo de respuesta. El aumento no proporcional indica problemas de escalabilidad.
  - Objetivo: buscar métricas y verificar si el rendimiento del sistema cambia adecuadamente a la carga.

Factores que afectan al rendimiento

- Si observamos características únicas del rendimiento del servicio web, se destacan dos aspectos: una gran cantidad de procesamiento XML / JSON en el lado del servidor (análisis y serialización XML / JSON) y procesamiento de cargas (cuerpos de solicitud y respuesta). Las razones por las cuales esto falla pueden ser múltiples; puede estar en la plataforma en forma de, por ejemplo, debilidades en el servidor de aplicaciones, y en la implementación en forma de WSDL innecesariamente complejos. También podría ser que el código está haciendo una solicitud a una base de datos que responde lentamente.
- Un factor más que afecta el rendimiento con frecuencia es la seguridad. Aquellos que realizan pruebas de rendimiento web saben que los sitios HTTPS tienen un rendimiento considerablemente menor que sus análogos HTTP, y en las pruebas de servicios web, podemos agregar una capa de WS-Security a la capa de seguridad HTTP, disminuyendo aún más el rendimiento.
- La complejidad de analizar las cargas útiles de XML / JSON significa que tenemos que poner un enfoque adicional en las pruebas de escalabilidad, mientras que el problema de la seguridad significa que tendremos que enfocarnos un poco más en hacer pruebas de solicitudes que son seguras. Si todo el servicio web es seguro, esto significa que la prueba de carga es más importante, especialmente si se utiliza WS-Security y el manejo de tokens.
- También significa que tenemos que examinar la especificación API de cerca. Si las solicitudes y respuestas son complejas o grandes, o si incluyen archivos adjuntos grandes, debemos centrarnos en enfatizar esa complejidad y ver cómo se comporta bajo carga.

LoadTest

- En SoapUI, los LoadTests se basan en los TestCases funcionales existentes. Un LoadTest ejecuta el TestCase repetidamente durante el tiempo especificado con un número deseado de subprocesos (o usuarios virtuales). Los LoadTests se muestran en el navegador como elementos secundarios del TestCase que usan. Se puede crear cualquier número de LoadTests para un TestCase.
- Las diferentes estrategias de carga disponibles en SoapUI permiten simular varios tipos de carga a lo largo del tiempo, lo que permite probar fácilmente el rendimiento de sus servicios de destino en una serie de condiciones. Dado que SoapUI también le permite ejecutar múltiples LoadTests simultáneamente, se puede usar una combinación de LoadTests para afirmar aún más el comportamiento de los servicios.
- Si queremos simular varios clientes consumiendo el servicio, vamos a la opción threads, por defecto aparecen 5 pero se puedan variar ese número a discreción, aunque mientras más hilos más consumo del CPU. Si queremos especificar una demora entre una petición y otra vamos a la opción delay (1000 milisegundos por defecto) y la cantidad aleatoria de la demora que se desea aleatorizar (es decir, establecerlo en 0.5 dará como resultado retrasos entre 0,5 y 1 segundo)
- Si queremos determinar el tiempo de duración del servicio vamos a la opción Limit y aquí se muestra que solo durará 60 segundos pero pueden cambiarla para especificar la cantidad de peticiones por hilo por ejemplo.

Ejecución de LoadTest

- SoapUI permite ejecutar el LoadTest con tantos subprocesos (o usuarios virtuales) como el hardware pueda administrar, dependiendo principalmente de la memoria, la CPU, el tiempo de respuesta del servicio objetivo y otros factores. Establezca el valor deseado e inicie LoadTest con el botón Ejecutar en la barra de herramientas del editor LoadTest.
- El TestCase subyacente se clona internamente para cada subproceso configurado y se inicia en su propio contexto; scripts, transferencias de propiedades, etc. El TestCase accederá a una "copia" única de TestCase y TestSteps que evita problemas de subprocesos en el nivel TestStep y TestCase (pero no más arriba en la jerarquía).
- A medida que se ejecuta LoadTest, la tabla de estadísticas de LoadTest se actualiza continuamente con los datos recopilados cada vez que finaliza un TestCase ejecutado, lo que le permite controlar de forma interactiva el rendimiento de sus servicios de destino mientras se ejecuta LoadTest. Si el TestCase contiene un bucle o TestSteps de ejecución prolongada, las estadísticas pueden tardar un tiempo en actualizarse (ya que se recopilan cuando finaliza TestCase), en este caso, seleccione la opción "Estadísticas TestStep" en el cuadro de diálogo Opciones de LoadTest para actualizarse estadísticas en el nivel TestStep en lugar del nivel TestCase (esto requiere un poco más de procesamiento interno, por eso está desactivado de forma predeterminada).

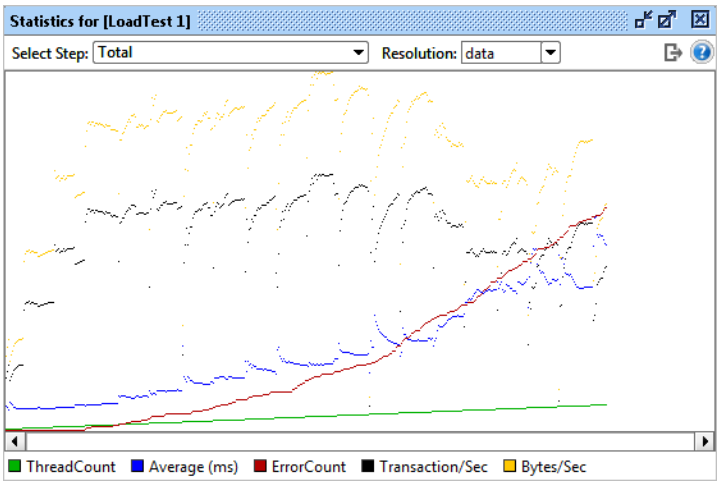
Estadísticas

- Tiempos mínimos (min), máximos (max), medios (avg) y de la última petición (last), todos en milisegundos.
- Número de ejecuciones (cnt), total de bytes (bytes), número errores (err) y el ratio de errores (rat).
- TPS (transacciones por segundo) y BPS (bytes por segundo) se pueden calcular de dos maneras diferentes (Opciones de LoadTest);
  - Según el tiempo real transcurrido (predeterminado):
    - TPS: cnt/segundos pasados, es decir, un TestCase que se ejecutó durante 10 segundos y manejó 100 solicitudes obtendrá un TPS de 10
    - BPS: bytes/tiempo transcurrido, es decir, un TestCase que se ejecutó durante 10 segundos y manejó 100000 bytes obtendrá un BPS de 10000.
  - Basado en el tiempo promedio de ejecución:
    - TPS: (1000/avg) * número de hilos, por ejemplo avg = 100 ms con diez hilos dará un TPS de 100
    - BPS: (bytes/cnt) * TPS, es decir, el número promedio de bytes por solicitud * TPS.
- Se pueden exportar a ficheros .csv

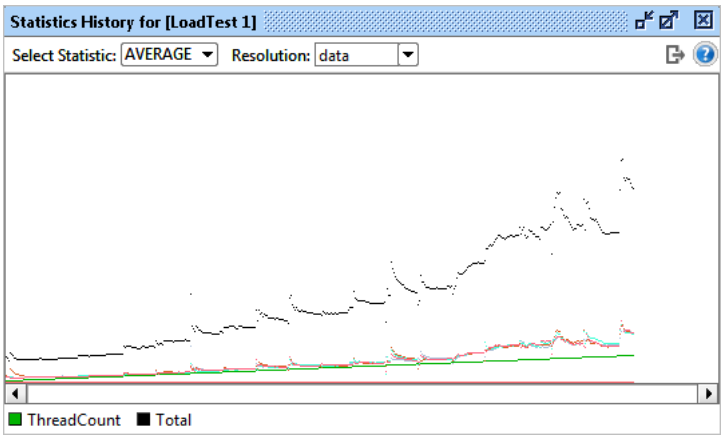
Gráficos estadísticos

- Hay dos tipos de gráficos disponibles en la barra de herramientas de LoadTest durante la ejecución; estadísticas e historial de estadísticas.
- El objetivo principal de estos es visualizar estadísticas seleccionadas a lo largo del tiempo para poder detectar cambios repentinos e inesperados. La visualización de estadísticas es relativa (no absoluta) y, por lo tanto, los gráficos no son muy útiles para analizar datos exactos.
- Ambos gráficos tienen una configuración de Resolución que controla con qué frecuencia se actualiza el gráfico; establecer esto en "datos" actualizará el gráfico con el mismo intervalo que la Tabla de estadísticas (que son los datos subyacentes para el gráfico, de ahí el nombre). Alternativamente, si desea una de las resoluciones fijas, seleccione estas.
- El gráfico de estadísticas muestra todas las estadísticas relevantes para un paso seleccionado o el caso de prueba completo a lo largo del tiempo, lo que le permite ver cómo cambian los valores cuando, por ejemplo, aumenta el número de subprocesos.
- El gráfico de historial de estadísticas muestra un valor estadístico seleccionado para todos los pasos que le permite compararlos y ver si la distribución de cualquier valor entre los pasos de prueba cambia con el tiempo.
- Sus datos se pueden exportar a ficheros .csv

Gráficos estadísticos



Gráficos estadísticos



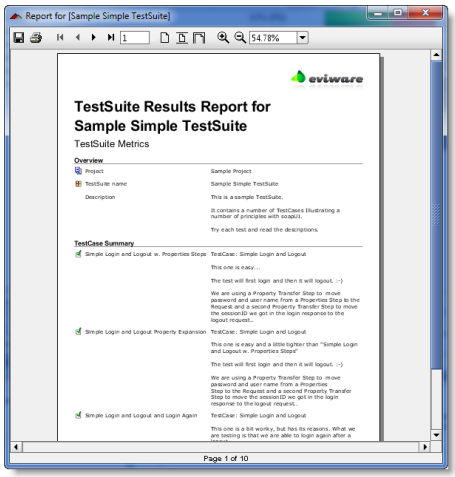
Validando el rendimiento

- Al igual que puede agregar aserciones funcionales a ciertos TestSteps en un TestCase (para afirmar sus resultados), se pueden agregar aserciones a un LoadTest para validar tanto el rendimiento como la funcionalidad durante la ejecución de LoadTest. Las aserciones configuradas se aplican continuamente a los resultados de LoadTest y puede fallar el LoadTest al igual que su contraparte funcional.
- Las aserciones disponibles son:
  - Step Average: valida que el valor promedio de un TestStep o todo el TestCase no excede el límite especificado.
  - Step TPS: valida el valor de TPS (transacción por segundo) para el TestStep o TestCase correspondiente.
  - Step Maximum: valida el valor tiempo máximo para el TestStep o TestCase correspondiente.
  - Step Status: comprueba que el estado de ejecución subyacente del TestStep o TestCase sea exitoso; de lo contrario, la aserción falla.
  - Max Errors: compruebe que el número de fallos para el TestStep o TestCase correspondiente no exceda el valor configurado.

Informes

- SoapUI Pro proporciona tres tipos de informes que se pueden generar desde el interior de la interfaz de usuario a nivel de Proyecto, TestSuite, TestCase y LoadTest:
  - Informes imprimibles: se pueden imprimir o guardar como PDF, HTML, RTF, Excel, etc. y son totalmente personalizables tanto a nivel global como de proyecto, lo que permite crear y personalizar cualquier tipo de informe que se pueda necesitar.
  - Exportación de datos: permite la exportación de datos de informes subyacentes en formato XML y CSV. Esto es útil si se desea importar datos de informes en otras herramientas para informes personalizados o integraciones.
  - Informes HTML: brinda una descripción general simplificada de los resultados de las pruebas funcionales en formato HTML (no disponible en el nivel de LoadTest).
- Se puede incluir fácilmente cualquier dato deseado en los informes generados, ya que puede exportar datos personalizados tanto en informes imprimibles como de exportación de datos a través del SubReport DataSink.

Informes



Data Driven Testing

- Las pruebas basadas en datos son aquellas que almacenan los datos de la prueba (entrada, salida esperada, etc.) en algún almacenamiento externo (base de datos, hoja de cálculo, archivos xml, etc.) y luego usa esos datos de forma iterativa en las pruebas cuando se ejecutan.
- Las pruebas basadas en datos solo están disponibles en SoapUI Pro y crearlas es realmente fácil.
- El DataSource TestStep permite la lectura de los datos de prueba en propiedades SoapUI estándar a partir de una serie de fuentes externas (archivos de Excel, propiedades XML, fuentes JDBC, archivos/directorios, etc.), estas propiedades se pueden utilizar en los siguientes TestSTEPS (solicitudes, aserciones, consultas XPath, scripts, etc.), ya sea a través de Transferencias de propiedad o Expansiones de propiedad. Finalmente un DataSource Loop TestStep permite recorrer la fuente de datos para aplicar los TestSteps anteriores para los datos de cada fila/registro del DataSource.
- Esto se reduce a la siguiente configuración:
  1. DataSource: lee datos de prueba en propiedades de una fuente externa
  2. TestSteps (cualquier número): prueba la funcionalidad del servicio utilizando las propiedades DataSource disponibles en (1)
  3. DataSource Loop: realiza un bucle de regreso a (2) para cada fila en (1)

TestRunner Command-Line

- Una vez creado, es posible que se desee ejecutar los Tests desde la línea de comandos, tal vez como parte de una compilación de integración continua o para monitorear el rendimiento diario de sus servicios.
- SoapUI proporciona utilidades de línea de comandos y complementos para que maven haga esto.
- Las utilidades están disponible en la carpeta SoapUI\bin (.bat o .sh):
  - testrunner.bat
  - loadtestrunner.bat
  - securitytestrunner.bat
  - mockservicerunner.bat
- Se necesitan muchos argumentos relacionados con informes, propiedades, etc., que pueden hacer que la creación de la lista de argumentos inicial sea bastante tediosa. Afortunadamente, la interfaz de usuario incluye un asistente para iniciar LoadTestRunner, que ayudará a evaluar la salida, así como a crear una cadena de invocación de línea de comandos.

https://www.soapui.org/test-automation/running-from-command-line/functional-tests.html

https://jmeter.apache.org/

JMETER

Introducción

- Apache JMeter es un software de código abierto, una aplicación Java 100% pura, diseñada para cargar el comportamiento funcional de las pruebas y medir el rendimiento.
- Originalmente fue diseñado para probar aplicaciones web, pero desde entonces se ha expandido a otras funciones de prueba.
- Apache JMeter puede usarse para probar el rendimiento tanto en recursos estáticos como dinámicos: aplicaciones web dinámicas.
- Se puede usar para simular una carga pesada en un servidor, grupo de servidores, red u objeto para probar su resistencia o para analizar el rendimiento general bajo diferentes tipos de carga.
- Jmeter no es apropiado para pruebas E2E dado que no es un navegador, funciona a nivel de protocolo. En lo que respecta a servicios web y servicios remotos, JMeter se parece a un navegador (o más bien, a múltiples navegadores); sin embargo, JMeter no realiza todas las acciones admitidas por los navegadores. En particular, JMeter no ejecuta el Javascript que se encuentra en las páginas HTML. Tampoco representa las páginas HTML como lo hace un navegador (es posible ver la respuesta como HTML, etc., pero los tiempos no se incluyen en ninguna muestra, y solo una muestra en una secuencia se muestra a la vez).

Características

- IDE de prueba con todas las funciones que permite la grabación rápida del plan de prueba (desde navegadores o aplicaciones nativas), construcción y depuración .
- Modo en línea de comandos para cargar la prueba desde cualquier SO compatible con Java (Linux, Windows, Mac OSX, ...)
- Un informe HTML completo y listo para presentar.
- Correlación sencilla mediante la capacidad de extraer datos de los formatos de respuesta más populares: HTML, JSON, XML o cualquier formato de texto
- Portabilidad completa y 100% Java puro.
- El marco completo de subprocesos múltiples permite el muestreo simultáneo mediante varios subprocesos y el muestreo simultáneo de diferentes funciones mediante grupos de subprocesos separados.
- Caché y análisis/reproducción fuera de línea de los resultados de las pruebas.

Características

- Núcleo altamente extensible:
  - Los muestreadores enchufables permiten capacidades de prueba ilimitadas.
  - Muestreadores de secuencias de comandos (lenguajes compatible con JSR223 como Groovy y BeanShell)
  - Se pueden elegir varias estadísticas de carga con temporizadores conectables .
  - Los complementos de análisis y visualización de datos permiten una gran extensibilidad y personalización.
  - Las funciones se pueden utilizar para proporcionar una entrada dinámica a una prueba o para proporcionar manipulación de datos.
  - Integración Continua simplificada a través de bibliotecas de código abierto de terceras partes para Maven, Gradle y Jenkins.

Pruebas de rendimiento

Pruebas de carga, estrés y picos

- Pruebas de carga (load test): pruebas para determinar y validar la respuesta de la aplicación cuando es sometida a una carga de usuarios y/o transacciones que se espera en el ambiente de producción. Ejemplo: verificar la correcta respuesta de la aplicación ante el alta de 100 usuarios en forma simultanea. Se compara con el volumen esperado.
- Pruebas de estrés (stress test): pruebas para encontrar el volumen de datos o de tiempo en que la aplicación comienza a fallar o es incapaz de responder a las peticiones. Son pruebas de carga o rendimiento, pero superando los límites esperados en el ambiente de producción y/o determinados en las pruebas. Ejemplo: encontrar la cantidad de usuarios simultáneos, en que la aplicación deja de responder (cuelgue o time out) en forma correcta a todas las peticiones.
- Pruebas de picos (spike testing): es un sub tipo de prueba de estrés que mide el rendimiento del software bajo un «pico» significativo y repentino o una carga de trabajo creciente como la de los usuarios simulados. Indica si el software puede manejar ese aumento abrupto de la carga de trabajo de forma repetida y rápida.

Pruebas de resistencia, volumen y escalabilidad

- Pruebas de resistencia (soak testing, endurance testing): evalúa el rendimiento del software durante un periodo prolongado bajo una carga de trabajo regular y fija, determina cuánto tiempo puede soportar el software una carga de trabajo constante para proporcionar sostenibilidad a largo plazo. Durante estas pruebas, los equipos de pruebas supervisan los KPI como las fugas de memoria, de proceso, etc. Las pruebas de resistencia también analizan los tiempos de respuesta y el rendimiento tras un uso prolongado para mostrar si estas métricas son consistentes.
- Pruebas de volumen (volume testing): comprueban la eficacia del software cuando se somete a grandes volúmenes de datos. Comprueba la pérdida de datos, el tiempo de respuesta del sistema, la fiabilidad del almacenamiento de datos, etc.
- Pruebas de escalabilidad (scalability testing): miden la eficacia del software a la hora de manejar una cantidad creciente de carga de trabajo, añadiendo volumen de datos o usuarios de forma gradual mientras se supervisa el rendimiento del software. La prueba informará sobre el comportamiento cuando aumenten o disminuyan los atributos de rendimiento del software.

Tipos de aplicaciones/servidores/protocolo

- Web: HTTP, HTTPS (Java, NodeJS, PHP, ASP.NET, ...)
- Servicios Web de SOAP / REST
- FTP
- Base de datos a través de JDBC
- LDAP
- Middleware orientado a mensajes (MOM) a través de JMS
- Correo: SMTP (S), POP3 (S) e IMAP (S)
- Comandos nativos o scripts de shell
- TCP
- Objetos Java

Debilidades

- Aunque permite también el diseño y automatización de pruebas funcionales, existen herramientas más adecuadas para ello.
- JMeter NO se comporta como un navegador. Esto tiene varias implicaciones importantes: por defecto no guarda ni envía cookies, no interpreta código JavaScript, … Cualquiera de estas funcionalidades debe ser implementada específicamente en el testplan
- Con JMeter el tester trabaja a nivel de protocolos: el desarrollador de un testplan ha de descender a este nivel, por lo que normalmente el tester tiene que apoyarse en herramientas adicionales durante el desarrollo de un testplan, como Firebug, HttpFox, SoapUI, Badboy, ...
- Los tipos de aplicaciones que se pueden testear con JMeter dependen de los protocolos que implementen las interfaces de acceso a la aplicación. Con JMeter se pueden testear los siguientes tipos de interfaces: HTTP, HTTPS, SOAP (sobre HTTP), XML-RPC (sobre HTTP), FTP, LDAP, POP3, IMAP, SMTP, JMS, JDBC y TCP. El testeo de otro tipo de interfaces no es inmediato. En concreto, con JMeter no es posible (con un esfuerzo razonable), implementar un testplan para una interfaz RMI o WebDav

Monitor de rendimiento de Windows

- El Monitor de rendimiento es la utilidad gráfica del sistema operativo que permite supervisar al sistema operativo Windows, principalmente para buscar cuellos de botella.
- Se basa en contadores de rendimiento asociados a los diferentes objetos del sistema, como procesadores, memoria, caché, subprocesos y procesos.
- Cada uno de estos objetos tiene asociado un conjunto de contadores que miden el uso de los dispositivos, la longitud de las colas, las demoras y otros indicadores del rendimiento y la congestión interna.

Contadores

- Disco:
  - PhysicalDisk: Avg. Disk Queue Length
  - PhysicalDisk: Avg. Disk sec/Read, PhysicalDisk: Avg. Disk sec/Write
  - Processor:% Privileged Time
- Procesador
  - Processor: % Processor Time
  - System: Processor Queue Length
- Memoria:
  - Memory: Page Faults/sec
  - Memory: Available Bytes
  - Memory: Pages/sec
- Red:
  - Network Interface: Bytes Total/sec
  - Network Interface: Current Bandwidth

Instalación

- Descargar Jmeter
  - https://jmeter.apache.org/
- Descomprimir
- Descargar JMeter Plugins
  - https://jmeter-plugins.org/install
- Copiar plugins-manager.jar en jmeter/lib/ext
- Configurar, si es necesario, jmeter/bin/jmeter.bat
- Abrir GUI: jmeter/bin/jmeterw.cmd

Modos

- Modo GUI
- Modo sin GUI (modo de línea de comandos)
  - Para las pruebas de carga, se debe ejecutar JMeter en modo sin GUI para obtener los resultados óptimos.

    jmeter -n -t my_test.jmx -l log.jtl
- Modo de servidor
  - Para las pruebas distribuidas, se ejecuta JMeter en modo servidor en los nodos remotos y se controla los servidores desde el GUI. También puede utilizar el modo no GUI para ejecutar pruebas remotas. Para iniciar cada servidor en cada host:

    jmeter-server

Elementos de un plan de prueba

- Grupo de hilos
- Controladores
- Receptores
- Temporizadores
- Aserciones
- Elementos de configuración
- Elementos del preprocesador
- Elementos de postprocesador
- Propiedades y variables
- Funciones

Grupo de hilos

- Los elementos del grupo de hilos son los puntos de inicio de cualquier plan de prueba. Todos los controladores y muestreadores deben estar bajo un grupo de subprocesos. El grupo de hilos controla el número de hilos que JMeter usará para ejecutar la prueba. Los controles para un grupo de hilos le permiten:
  - Establecer el número de hilos
  - Establecer el período de aceleración
  - Establecer el número de veces para ejecutar la prueba.
- Cada hilo ejecutará el plan de prueba en su totalidad y de forma completamente independiente de otros hilos de prueba. Se utilizan varios subprocesos para simular conexiones simultáneas a su aplicación de servidor.
- El período de aceleración le dice a JMeter cuánto tiempo se tarda en "aumentar" la cantidad total de hilos elegidos.
- La expansión debe ser lo suficientemente larga para evitar una carga de trabajo demasiado grande al inicio de una prueba, y lo suficientemente corta para que los últimos hilos comiencen a ejecutarse antes de que terminen los primeros (a menos que uno quiera que eso ocurra).
- Así mismo permite controlar la acción a tomar después de un error del muestrador.

Controladores

- Los controladores conducen el procesamiento de una prueba. JMeter tiene dos tipos de controladores: muestreadores y controladores lógicos.
- Los muestreadores le dicen a JMeter que envíe solicitudes a un servidor. Por ejemplo, se agrega un muestreador de Petición HTTP si se desea que JMeter envíe una solicitud HTTP. También se puede personalizar una solicitud agregando uno o más elementos de configuración a un muestreador.
- Los controladores lógicos permiten personalizar la lógica que JMeter utiliza para decidir cuándo enviar solicitudes. Por ejemplo, puede agregar un controlador de lógica de intercalación para alternar entre dos muestreadores de solicitud HTTP.
- El Fragmento de prueba es un tipo especial de controlador que existe en el árbol del Plan de prueba al mismo nivel que el Grupo de subprocesos. Se distingue de un Grupo de subprocesos en que no se ejecuta a menos que sea referenciado por un Controlador de Módulo o de Include. Este elemento es únicamente para reutilizar el código dentro de los planes de prueba.

Muestreadores

- Los muestreadores le dicen a JMeter que envíe solicitudes a un servidor y espere una respuesta. Se procesan en el orden en que aparecen en el árbol. Los controladores se pueden utilizar para modificar el número de repeticiones de una muestra.
- Los muestreadores JMeter incluyen:
  - Solicitud HTTP (también se puede utilizar para el servicio web SOAP o REST)
  - Solicitud de FTP
  - Solicitud de correo
  - Solicitud de TCP
  - Solicitud de objeto Java
  - Solicitud JDBC
  - Solicitud JMS
  - Solicitud de prueba de JUnit
  - Solicitud LDAP
  - Solicitud de proceso OS

Controladores lógicos

- Los controladores lógicos le permiten personalizar la lógica que JMeter utiliza para decidir cuándo enviar solicitudes. Los controladores lógicos pueden cambiar el orden de las solicitudes provenientes de sus elementos secundarios. Pueden modificar las solicitudes ellos mismos, hacer que JMeter repita las solicitudes, etc.
- Mediante los controladores lógicos se pueden establecer transacciones, orden, frecuencia y el flujo de control con condicionales, bucles, …

Logic Controllers

- Simple Controller
- Loop Controller
- Once Only Controller
- Interleave Controller
- Random Controller
- Random Order Controller
- Throughput Controller
- Runtime Controller
- If Controller
- While Controller
- Switch Controller
- ForEach Controller
- Module Controller
- Include Controller
- Transaction Controller
- Recording Controller
- Critical Section Controller

Receptores

- Los receptores proporcionan acceso a la información que JMeter recopila sobre los casos de prueba mientras JMeter se ejecuta. La mayoría de los receptores realizan varias funciones además de "escuchar" los resultados de la prueba. También proporcionan medios para ver, guardar y leer los resultados de las pruebas guardadas.
- Se pueden agregar receptores en cualquier parte de la prueba, incluso directamente en el plan de prueba. Recopilarán datos solo de elementos en el que están asociado o por debajo de su nivel.
- Todos los receptores guardan los mismos datos, la única diferencia está en la forma en que se presentan los datos en pantalla.
- Los receptores pueden dirigir los datos a un archivo para su uso posterior, proporcionan un campo para indicar el archivo para almacenar datos y permiten elegir qué campos guardar así como el formato: CSV o XML.
- Los receptores pueden presentar los datos en formato grafico, tabular, árbol, especializado y para depurar. Algunos proporcionan información de resumen o agregación.

Listeners

- Sample Result Save Configuration
- Graph Results
- Assertion Results
- View Results Tree
- Aggregate Report
- View Results in Table
- Simple Data Writer
- Aggregate Graph
- Response Time Graph
- Mailer Visualizer
- BeanShell Listener
- Summary Report
- Save Responses to a file
- JSR223 Listener
- Generate Summary Results
- Comparison Assertion Visualizer
- Backend Listener

Temporizadores

- De forma predeterminada, un hilo de JMeter ejecuta los muestreadores en secuencia sin pausa. Es recomendable especificar un retraso agregando uno de los temporizadores disponibles al grupo de subprocesos, si no, JMeter podría saturar al servidor al realizar demasiadas solicitudes en un período de tiempo muy corto.
- Un temporizador hará que JMeter demore un cierto tiempo antes de cada muestra que esté dentro de su alcance.
- Si se agrega más de un temporizador a un grupo de subprocesos, JMeter toma la suma de los temporizadores y se detiene durante ese tiempo antes de ejecutar los muestreadores a los que se aplican los temporizadores. Los temporizadores se pueden agregar como hijos de muestreadores o controladores para restringir a los que se aplican.
- Para proporcionar una pausa en un solo lugar en un plan de prueba, se puede usar el Muestreador de acciones de prueba.

Timers

- Constant Timer
- Gaussian Random Timer
- Uniform Random Timer
- Constant Throughput Timer
- Precise Throughput Timer
- Synchronizing Timer
- BeanShell Timer
- JSR223 Timer
- Poisson Random Timer

Aserciones

- Las aserciones permiten afirmar hechos sobre las respuestas recibidas del servidor que se está probando, en caso de no cumplirse JMeter lo marcará como una solicitud fallida. Permiten comprobar que la aplicación está devolviendo los resultados esperados.
- Se pueden establecer aserciones sobre el contenido, tamaño y duración de las respuestas. Hay disponibles aserciones especificas para determinados muestreadores.
- Las aserciones se aplican a todos los muestreadores que se encuentran en su alcance. Para restringir una aserción a un solo muestreador se le debe agregar como elemento secundario.
- Para ver los resultados de la aserción, se agrega un receptor de resultado de aserción. Las aserciones fallidas también se mostrarán en la vista de árbol y en los receptores de la tabla, y contarán para el % de error en los informes de agregados y resumen.

Assertions

- Response Assertion
- Duration Assertion
- Size Assertion
- XML Assertion
- BeanShell Assertion
- MD5Hex Assertion
- HTML Assertion
- XPath Assertion
- XPath2 Assertion
- XML Schema Assertion
- JSR223 Assertion
- Compare Assertion
- SMIME Assertion
- JSON Assertion
- JSON JMESPath Assertion

Elementos de configuración

- Los elementos de configuración se pueden usar para configurar valores predeterminados y variables para su uso posterior por parte de los muestreadores. Hay que tener en cuenta que estos elementos se procesan al inicio del alcance en el que se encuentran, es decir, antes de cualquier muestreador en el mismo alcance.
- Están disponibles configuradores para:
  - Valores predeterminados para peticiones HTTP, FTP, Java, LDAP
  - Configuración de conexión JDBC, del almacén de claves, de muestra de TCP, de inicio de sesión
  - Variables definidas por el usuario, aleatorias, contadores, conjuntos de datos CSV
  - Administrador de caché HTTP y DNS
  - Administrador de autorización, cookies y encabezados HTTP

Configuration Elements

- CSV Data Set Config
- FTP Request Defaults
- DNS Cache Manager
- HTTP Authorization Manager
- HTTP Cache Manager
- HTTP Cookie Manager
- HTTP Request Defaults
- HTTP Header Manager
- Java Request Defaults
- JDBC Connection Configuration
- Keystore Configuration
- Login Config Element
- LDAP Request Defaults
- LDAP Extended Request Defaults
- TCP Sampler Config
- User Defined Variables
- Random Variable
- Counter
- Simple Config Element
- MongoDB Source Config (DEPRECATED)
- Bolt Connection Configuration

Elementos del preprocesador

- Un preprocesador ejecuta alguna acción antes de que se realice una solicitud de muestra.
- Si se adjunta un preprocesador a un elemento muestreador, entonces se ejecutará justo antes de que se ejecute ese elemento.
- Un preprocesador se utiliza principalmente para modificar la configuración de una solicitud de muestra justo antes de que se ejecute, o para actualizar las variables que no se extraen del texto de respuesta.

Elementos del postprocesador

- Un postprocesador ejecuta alguna acción después de que se haya realizado una solicitud de muestra.
- Si un postprocesador está conectado a un elemento Sampler, se ejecutará justo después de que se ejecute ese elemento.
- Un postprocesador se utiliza para procesar los datos de respuesta, a menudo para extraer valores de él.

Processors

Pre Processors

- HTML Link Parser
- HTTP URL Re-writing Modifier
- User Parameters
- BeanShell PreProcessor
- JSR223 PreProcessor
- JDBC PreProcessor
- RegEx User Parameters
- Sample Timeout

Post Processors

- Regular Expression Extractor
- CSS Selector Extractor (was: CSS/JQuery Extractor)
- XPath2 Extractor
- XPath Extractor
- JSON JMESPath Extractor
- Result Status Action Handler
- BeanShell PostProcessor
- JSR223 PostProcessor
- JDBC PostProcessor
- JSON Extractor
- Boundary Extractor

Propiedades y variables

- Las propiedades son globales para jmeter, y se usan principalmente para definir algunos de los valores predeterminados que usa JMeter. Por ejemplo, la propiedad remote_hosts define los servidores que JMeter intentará ejecutar de forma remota. Se puede hacer referencia a las propiedades en los planes de prueba pero no se puede usar para valores específicos de hilos. Las propiedades de JMeter se definen en los ficheros jmeter.properties y user.properties o en la línea de comandos.
- Las variables de JMeter son locales para cada hilo. Los valores pueden ser los mismos para cada hilo, o pueden ser diferentes. Si una variable es actualizada por un hilo, solo se cambia la copia de la variable en el hilo. Por ejemplo, el Postprocesador de Expresión Regular Extractor establecerá sus variables de acuerdo con la muestra que su hilo haya leído, y estas pueden usarse más adelante por el mismo hilo.
- Los valores definidos por el plan de prueba y el configurador de variables definidas por el usuario (UDV) están disponibles para todo el plan de prueba desde el inicio. Si la misma variable está definida por múltiples UDV, solo tendrá efecto la última.

Propiedades y variables

- Una vez que un hilo ha comenzado, el conjunto inicial de variables se copia a cada hilo. Otros elementos, como el Preprocesador de parámetros de usuario o el Postprocesador de extracción de expresiones regulares, se pueden usar para redefinir las mismas variables (o crear nuevas). Estas redefiniciones solo se aplican al hilo actual.
- Las variables no tienen por que variar, se pueden definir una vez y, si se dejan solas, no cambiarán de valor. Por lo tanto, se pueden usar como abreviatura para expresiones que aparecen con frecuencia en un plan de prueba o para los elementos que son constantes durante una ejecución pero que pueden variar entre ejecuciones. Por ejemplo, el nombre de un host.
- Tanto las variables como las propiedades distinguen entre mayúsculas y minúsculas.

Funciones

- Las funciones de JMeter son valores calculados que pueden rellenar los campos de cualquier muestreador u otro elemento en un árbol de prueba.
- Hay dos tipos de funciones: valores estáticos definidos por el usuario (o variables) y funciones incorporadas.
- Los valores estáticos definidos por el usuario permiten al usuario definir las variables que se reemplazarán con su valor estático cuando se compila un árbol de prueba y se envía para que se ejecute. Este reemplazo ocurre una vez al comienzo de la prueba.
- Con las funciones integradas, se pueden calcular nuevos valores en tiempo de ejecución dependiendo de los datos de respuesta anteriores, en qué hilo se encuentra la función, la hora y muchas otras fuentes. Se generan nuevos valores para cada solicitud a lo largo del curso de la prueba.

Funciones

- Las funciones y variables se pueden escribir en cualquier campo de cualquier componente de prueba.
- Para invocar las funciones:
  ${__functionName (var1, var2, var3)}
- Si un parámetro de función contiene una coma, entonces asegúrese de escapar con " \ ", de lo contrario JMeter lo tratará como un delimitador de parámetros. Las funciones que no requieren parámetros pueden omitir los paréntesis.
- Las variables se referencian de la siguiente manera:
  ${VARIABLE}
- Las propiedades deben ser referenciadas utilizando la función __P o __property:
  ${__property(user.dir)}
- Si se hace referencia a una función o variable no definida, JMeter NO informa/registra un error: la referencia se devuelve sin cambios.

Orden de ejecución

1. Elementos de configuración
2. Preprocesadores
3. Temporizadores
4. Controladores
5. Postprocesadores (si SampleResult no es nulo)
6. Aserciones (si SampleResult no es nulo)
7. Receptores (si SampleResult no es nulo)

   – Los temporizadores, las aserciones, los preprocesadores y los postprocesadores solo se procesan si hay un muestreador al que aplicarlo.
   – Los controladores lógicos y los muestreadores se procesan en el orden en que aparecen en el árbol.
   – Otros elementos de prueba se procesan de acuerdo con el alcance en el que se encuentran y el tipo de elemento de prueba.

Reglas de alcance

- El árbol de la prueba JMeter contiene elementos que son jerárquicos y están ordenados. Algunos elementos en los árboles de prueba son estrictamente jerárquicos (receptores, elementos de configuración, postprocesadores, preprocesadores, aserciones, temporizadores) y algunos están ordenados por principio (controladores, muestreadores).
- Cuando se crea el plan de prueba, se creará una lista ordenada de solicitudes de muestra (a través de muestreadores) que representa un conjunto de pasos a ejecutar. Estas solicitudes a menudo se organizan dentro de los controladores que también se ordenan.
- Otros elementos son jerárquicos. Una aserción, por ejemplo, es jerárquica en el árbol de prueba. Si su padre es una solicitud, entonces se aplica a esa solicitud. Si su padre es un controlador, afecta a todas las solicitudes que contiene.

Proceso de Prueba

- Preguntas iniciales¶
  - ¿Cuál es el número promedio anticipado de usuarios (carga normal)?
  - ¿Cuál es el número máximo de usuarios previsto?
  - ¿Cuándo es un buen momento para cargar y probar la aplicación (es decir, fuera de horario o fines de semana), teniendo en cuenta que esto puede muy bien bloquear uno o más de los servidores?
  - ¿La aplicación tiene estado? Si es así, ¿cómo la administra la aplicación (cookies, reescritura de URL o algún otro método)?
  - ¿Cuál es el objetivo que se pretende lograr con la prueba?
  - ¿Quién sabe cómo funciona la aplicación?
- La secuencia normal es:
  - Prueba (volumen bajo, ¿podemos comparar nuestra aplicación?)
  - Rendimiento (el número promedio de usuarios)
  - Carga (el número máximo de usuarios)
  - Estrés (¿cuál es el límite?)
- El proceso de prueba puede pasar de la prueba de caja negra a la prueba de caja blanca (la diferencia es que la primera no requiere conocimiento de la aplicación [se trata como una "caja negra"] mientras que la segunda requiere algún conocimiento de la aplicación). No es raro que se descubran problemas con la aplicación durante este proceso, así que prepárate para defender tu trabajo.

Prueba de carga

- En el GUI:
  - Crear y Validar Plan de Pruebas
  - Quitar/deshabilitar Receptores
  - Cerrar
- Ejecutar el plan en modo comando y exportar resultados:
  - jmeter –n *-f* -t demo.jmx -l demo.jtl -e -o demoinf
    - Sumario cada 30 segundos, totalizados
- Analizar resultados en el GUI:
  - Crear plan con receptores.
  - Cargar los ficheros de resultados
  - Ver informe HTML de resultados

Pruebas distribuidas

- En los sistemas esclavos, ejecutar jmeter/bin/ejecute jmeter-server.bat (jmeter-server en unix).
- En el sistema maestro, modificar el fichero de propiedades jmeter/bin/jmeter.properties:

  remote_hosts=192.168.0.10,192.168.0.11,192.168.0.12
- En el sistema maestro, iniciar Jmeter y abrir el plan de prueba que se desea utilizar.
- Para ejecutar la prueba:
  - Iniciar un solo cliente: Run > Remote Start > 192.168.0.10
  - Iniciar todos los clientes: Run > Remote Start All
- Para detener los sistemas esclavos, Run > Remote Stop > 192.168.0.10 ó Run > Remote Stop All

Informes y Análisis

- Apache JMeter Dashboard
  - Generación después de la prueba de carga:

    jmeter -n -t <archivo de prueba JMX> -l <archivo de registro de prueba> -e -o <Ruta a la carpeta de salida>
  - Generación a partir de un archivo de registro CSV de muestra existente

    jmeter -g <archivo de registro> -o <Ruta a la carpeta de salida>
- Respondedores

Respondedores de informe

- El receptor de resultados gráficos genera un gráfico simple que representa todos los tiempos de muestra. En la parte inferior del gráfico, la muestra actual (negro), el promedio actual de todas las muestras (azul), la desviación estándar actual (rojo) y la tasa de rendimiento actual (verde) se muestran en milisegundos.
- El informe agregado crea una fila de tabla para cada solicitud con un nombre diferente en la prueba. Para cada solicitud, totaliza la información de respuesta y proporciona el recuento de solicitudes, mínimo, máximo, promedio, tasa de error, rendimiento aproximado (solicitud / segundo) y rendimiento de Kilobytes por segundo. Una vez que se realiza la prueba, el rendimiento es el real a lo largo de toda la prueba.
- El gráfico agregado es similar al informe agregado. La principal diferencia es que el gráfico agregado proporciona una manera fácil de generar gráficos de barras exportables a PNG.
- El informe de resumen crea una fila de tabla para cada solicitud con un nombre diferente en su prueba. Similar al Informe agregado, pero usando menos memoria.
- El gráfico de tiempo de respuesta dibuja un gráfico de líneas que muestra la evolución del tiempo de respuesta durante la prueba, para cada solicitud etiquetada. Si existen muchas muestras para la misma marca de tiempo, se muestra el valor medio.

CI INSTALACIÓN (JENKINS Y SONAR)

Contenedores

- Crear versión Jenkins con Maven. Crear fichero llamado Dockerfile:
  FROM jenkins/jenkins:lts
  USER root
  RUN apt-get update && apt-get install -y maven
- Generar imagen Docker:
  - docker build -t jenkins-maven .
- Crear contenedores Docker
  - docker run -d --name sonarQube -p 9000:9000 sonarqube:latest
  - docker run -p 20000:8080 -p 50000:50000 -v D:\Cursos\Docker\jenkins_home:/var/jenkins --name jenkins-maven jenkins-maven
    - Copiar clave generada para proceder a la instalación (/var/jenkins_home/secrets/initialAdminPassword)

Red e Instalación de Jenkins

- Configurar red docker
  - docker network create red_jenkins_sonarqube
  - docker network connect red_jenkins_sonarqube jenkins-maven
  - docker network connect red_jenkins_sonarqube sonarQube
- En el navegador: http://localhost:20000/
  - Desbloquear Jenkins con la clave copiada
  - Install suggested plugins
  - Create First Admin User
  - Instance Configuration Jenkins URL: http://localhost:20000
  - Save and Finish
  - Start using Jenkins

Instalación de Plugins en Jenkins

- Administrar Jenkins > Instalar Plugins
  - JaCoCo
  - Html Publisher
  - SonarQube Scanner
  - Pipeline Maven Integration
  - Re arrancar al terminar
- SonarQube: http://loaclhost:9000
  - Usuario: admin
  - Contraseña: admin
  - Generar un token en Usuario > My Account > Security
    - Generate Tokens: Jenkins
    - Generate
    - Copiar

Configuración de Plugins en Jenkins

- Administrar Jenkins > Manage Credentials
  - Click link (global) in System table Global credentials (unrestricted).
  - Click Add credentials
    - Kind: Secret Text
    - Scope: Global
    - Secret: Pegar el token generado en SonarQube
    - Description: Sonar Token
- Administrar Jenkins > Configurar el Sistema
  - En SonarQube servers:
    - Add SonarQube
    - Name: sonarQube (estrictamente el mismo de la instalación docker)
    - URL del servidor: http://sonarQube:9000
    - Server authentication token: Sonar Token
    - Guardar

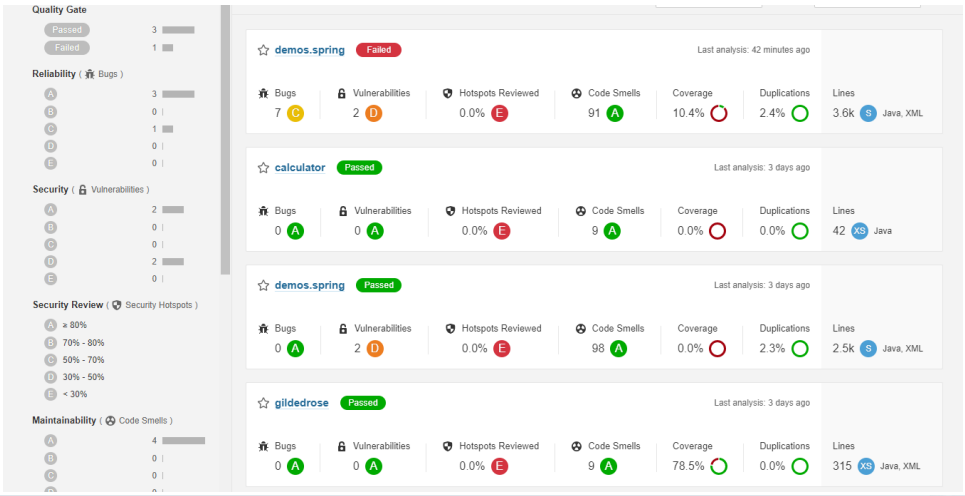
http://www.sonarqube.org/

SONARQUBE

Introducción

- SonarQube (conocido anteriormente como Sonar) es una herramienta de revisión automática de código para detectar errores, vulnerabilidades y malos olores en su código. Puede integrarse con su flujo de trabajo existente para permitir la inspección continua de código en las ramas de su proyecto y las solicitudes de incorporación de cambios.
- SonarQube es una plataforma para la revisión y evaluación del código fuente. Es una herramienta esencial para la fase de pruebas y auditoría de código dentro del ciclo de desarrollo de una aplicación y se considera perfecta para guiar a los equipos de desarrollo durante las revisiones de código.
- Es open source y realiza el análisis estático de código fuente integrando las mejores herramientas de medición de la calidad de código como Checkstyle, PMD o FindBugs, para obtener métricas que pueden ayudar a mejorar la calidad del código de un programa.
- Informa sobre código duplicado, estándares de codificación, pruebas unitarias, cobertura de código, complejidad ciclomática, posible errores, comentarios y diseño del software.
- Aunque pensado para Java, acepta mas de 20 lenguajes mediante extensiones. Se integra con Maven, Ant y herramientas de integración continua como Atlassian, Jenkins y Hudson.

Sonar



Proceso

1. Los desarrolladores desarrollan y combinan código en un IDE (preferiblemente usando SonarLint para recibir comentarios inmediatos en el editor) y registran su código en su plataforma DevOps.
2. La herramienta de integración continua (CI) de una organización verifica, crea y ejecuta pruebas unitarias, y un escáner SonarQube integrado analiza los resultados.
3. El escáner publica los resultados en el servidor de SonarQube, que proporciona comentarios a los desarrolladores a través de la interfaz de SonarQube, correo electrónico, notificaciones en el IDE (a través de SonarLint) y decoración en las solicitudes de extracción o combinación (cuando se usa Developer Edition y superior).

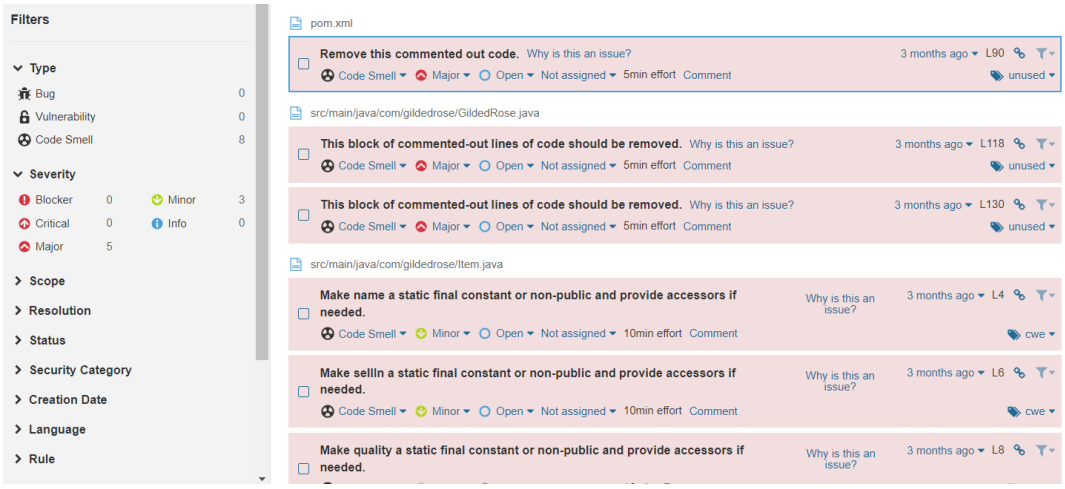
Beneficios

- Alerta de manera automática a los desarrolladores de los errores de código para corregirlos previamente a la implementación en producción.
- No sólo muestra los errores, también las reglas de codificación, la cobertura de las pruebas, las duplicaciones, la complejidad y la arquitectura, plasmando todos estos datos en paneles de control detallados.
- Ayuda al equipo a mejorar en sus habilidades como programadores al facilitar un seguimiento de los problemas de calidad.
- Permite la creación de paneles y filtros personalizables para centrarse en áreas clave y entregar productos de calidad a tiempo.
- Favorece la productividad al reducir la complejidad del código acortando tiempos y costes adicionales al evitar cambiar el código constantemente.

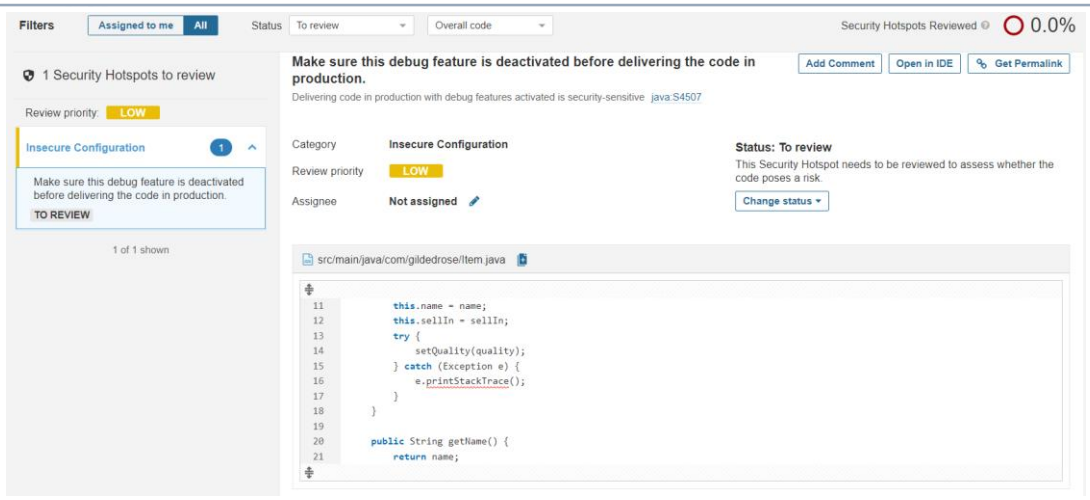
Herramientas

- SonarLint: es un producto complementario que funciona en su editor y brinda comentarios inmediatos para que pueda detectar y solucionar problemas antes de que lleguen al repositorio.
- Quality Gate: permite saber si su proyecto está listo para la producción.
- Clean as You Code: es un enfoque de la calidad del código que elimina muchos de los desafíos que conllevan los enfoques tradicionales. Como desarrollador, se enfoca en mantener altos estándares y asumir la responsabilidad específicamente en el Nuevo Código en el que está trabajando.
- Issues: SonarQube plantea problemas cada vez que una parte de su código infringe una regla de codificación, ya sea un error que romperá su código (bug), un punto en su código abierto a ataques (vulnerabilidad) o un problema de mantenimiento (código apestoso).
- Security Hotspots: Puntos de acceso de seguridad destaca piezas de código sensibles a la seguridad que deben revisarse. Tras la revisión, descubrirá que no hay ninguna amenaza o que debe aplicar una solución para proteger el código.

Problemas



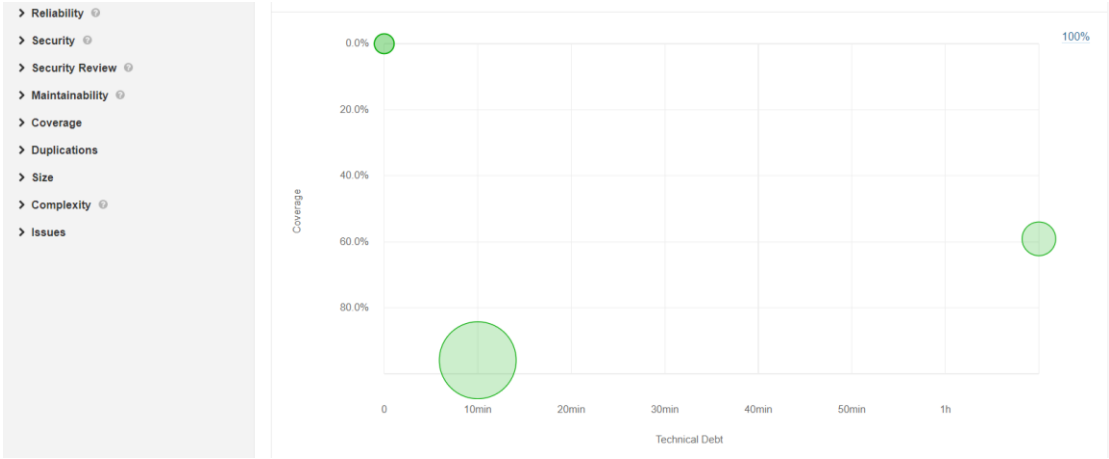
Puntos de acceso de seguridad



Métricas principales

- **Complejidad**: Refleja la Complejidad Ciclomática calculada en base al número de caminos a través del código normalmente observado a nivel de métodos o funciones individuales.
- **Duplicados**: Nos indica el número de bloques de líneas duplicados. Ayuda a evitar resultados distintos en operaciones iguales.
- **Evidencias**: Son los fragmentos nuevos de código de un proyecto que detecta que incumplen con alguna de las reglas establecidas.
- **Mantenibilidad**: Se refiere al recuento total de problemas de Code Smell.
- **Umbrales de calidad**: Define los requisitos del proyecto antes de ser lanzado a producción, como, por ejemplo, que no deben haber evidencias bloqueantes o la cobertura de código sobre el código nuevo debe ser mayor que el 80%.
- **Tamaño**: Permiten hacerse una idea del volumen del proyecto en términos generales.
- **Pruebas**: Son una forma de comprobar el correcto funcionamiento de una unidad de código y de su integración.

Métricas



Análisis

- SonarQube puede analizar mas de 20 lenguajes diferentes según la edición. El resultado de este análisis serán métricas y problemas de calidad (casos en los que se rompieron las reglas de codificación). Sin embargo, lo que se analiza variará según el lenguaje:
  - En todos los lenguajes, los datos de "culpa" se importarán automáticamente de los proveedores de SCM admitidos. Git y SVN son compatibles automáticamente.
  - En todos los lenguajes se realiza un análisis estático del código fuente (archivos Java, programas COBOL, etc.)
  - Se puede realizar un análisis estático del código compilado para ciertos lenguajes (archivos .class en Java, archivos .dll en C#, etc.)
- Durante el análisis, se solicitan datos del servidor, se analizan los archivos proporcionados para el análisis enfrentándolos a las reglas y los datos resultantes se envían de vuelta al servidor al final en forma de informe, que luego se analiza de forma asíncrona en el lado del servidor.

Reglas

- Una regla es un estándar o práctica de codificación que debe seguirse. El incumplimiento de las reglas de codificación conduce a errores, vulnerabilidades, puntos críticos de seguridad y código apestoso. Las reglas pueden comprobar la calidad de los archivos de código o las pruebas unitarias.
- Cada lenguaje de programación dispone de sus propias reglas, siendo diferentes en cantidad y tipo.
- Las reglas se clasifican en:
  - Bug: un punto de fallo real o potencial en su software
  - Code Smell: un problema relacionado con la mantenibilidad en el código.
  - Vulnerability: un punto débil que puede convertirse en un agujero de seguridad que puede usarse para atacar su software.
  - Security Hotspot: un problema que es un agujero de seguridad.
- Para Bugs y Code Smells and Bugs, no se esperan falsos positivos. Al menos este es el objetivo para que los desarrolladores no tengan que preguntarse si se requiere una solución. Para las vulnerabilidades, el objetivo es que más del 80 % de los problemas sean verdaderos positivos. Las reglas de Security Hotspot llaman la atención sobre el código que es sensible a la seguridad. Se espera que más del 80% de los problemas se resuelvan rápidamente como "Revisados" después de la revisión por parte de un desarrollador.

Control de calidad

- Los perfiles de calidad (Quality Profile) son un componente central de SonarQube donde define conjuntos de reglas que, cuando se violan, generan problemas en su base de código (ejemplo: los métodos no deben tener una complejidad cognitiva superior a 15). Cada lenguaje individual tiene su propio perfil de calidad predeterminado y los proyectos que no están asignados explícitamente a perfiles de calidad específicos se analizan utilizando los perfiles de calidad predeterminados.
- Una barrera de calidad (Quality Gate) es un conjunto de condiciones booleanas basadas en métricas. Le ayuda a saber inmediatamente si sus proyectos están listos para la producción. Idealmente, todos los proyectos utilizarán la misma barrera de calidad. El estado de Quality Gate de cada proyecto se muestra de forma destacada en la página de inicio.

Scanner

- Una vez que se haya instalado la plataforma SonarQube, estará listo para instalar un escáner y comenzar a crear proyectos. Para ello, se debe instalar y configurar el escáner más adecuado pare el proyecto:
  - Gradle - SonarScanner para Gradle
  - Ant - SonarScanner para Ant
  - Maven: use el SonarScanner para Maven
  - .NET - SonarScanner para .NET
  - Jenkins - SonarScanner para Jenkins
  - Azure DevOps - Extensión de SonarQube para Azure DevOps
  - Cualquier otra cosa (CLI) - SonarScanner

SonarScanner para Maven

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.10.0</version>
        </plugin>
        <plugin>
            <groupId>org.sonarsource.scanner.maven</groupId>
            <artifactId>sonar-maven-plugin</artifactId>
            <version>3.9.1.2184</version>
        </plugin>
        <plugin>
            <groupId>org.jacoco</groupId>
            <artifactId>jacoco-maven-plugin</artifactId>
            <version>0.8.7</version>
        </plugin>
    </plugins>
</build>
```

SonarScanner para Maven

```
<profiles>
	<profile>
		<id>coverage</id>
		<activation>
			<activeByDefault>true</activeByDefault>
		</activation>
		<build>
			<plugins>
				<plugin>
					<groupId>org.jacoco</groupId>
					<artifactId>jacoco-maven-plugin</artifactId>
					<executions>
						<execution>
							<id>prepare-agent</id>
							<goals>
								<goal>prepare-agent</goal>
							</goals>
						</execution>
						<execution>
							<id>report</id>
							<goals>
								<goal>report</goal>
							</goals>
						</execution>
					</executions>
				</plugin>
			</plugins>
		</build>
	</profile>
</profiles>
```


Analizando

- Analizar un proyecto Maven consiste en ejecutar un objetivo Maven: sonar:sonar desde el directorio que contiene el proyecto principal pom.xml.
- Se debe pasar un token de autenticación usando la propiedad sonar.login en la línea de comando.
  - mvn sonar:sonar -Dsonar.projectKey=Microservicios -Dsonar.host.url=http://localhost:9000 -Dsonar.login=1eeaaf436541...
- Para obtener información de cobertura, deberá generar el informe de cobertura antes del análisis.
  - mvn clean verify sonar:sonar

JENKINS

Introducción

- Jenkins es un servidor open source para la integración continua. Es una herramienta que se utiliza para compilar y probar proyectos de software de forma continua, lo que facilita a los desarrolladores integrar cambios en un proyecto y entregar nuevas versiones a los usuarios. Escrito en Java, es multiplataforma y accesible mediante interfaz web. Es el software más utilizado en la actualidad para este propósito.
- Con Jenkins, las organizaciones aceleran el proceso de desarrollo y entrega de software a través de la automatización. Mediante sus centenares de plugins, se puede implementar en diferentes etapas del ciclo de vida del desarrollo, como la compilación, la documentación, el testeo o el despliegue.
- La primera versión de Jenkins surgió en 2011, pero su desarrollo se inició en 2004 como parte del proyecto Hudson. Kohsuke Kawaguchi, un desarrollador de Java que trabajaba en Sun Microsystems, creó un servidor de automatización para facilitar las tareas de compilación y de realización de pruebas.
- En 2010, surgieron discrepancias relativas a la gestión del proyecto entre la comunidad y Oracle y, finalmente, se decidió el cambio de denominación a “Jenkins”. Desde entonces los dos proyectos continuaron desarrollándose independientemente. Hasta que finalmente Jenkins se impuso al ser utilizado en muchos más proyectos y contar con más contribuyentes.

Por qué usarlo

- Antes de disponer de herramientas como Jenkins para poder aplicar integración continua nos encontrábamos con un escenario en el que:
  - Todo el código fuente era desarrollado y luego testeado, con lo que los despliegues y las pruebas eran muy poco habituales y localizar y corregir errores era muy laborioso. El tiempo de entrega del software se prolongaba.
  - Los desarrolladores tenían que esperar al desarrollo de todo el código para poner a prueba sus mejoras.
  - El proceso de desarrollo y testeo eran manuales, por lo que era más probable que se produjeran fallos.
- Sin embargo, con Jenkins (y la integración continua que facilita) la situación es bien distinta:
  - Cada commit es desarrollado y verificado. Con lo que en lugar de comprobar todo el código, los desarrolladores sólo necesitan centrarse en un commit concreto para corregir bugs.
  - Los desarrolladores conocen los resultados de las pruebas de sus cambios durante la ejecución.
  - Jenkins automatiza las pruebas y el despliegue, lo que ahorra mucho tiempo y evita errores.
  - El ciclo de desarrollo es más rápido. Se entregan más funcionalidades y más frecuentemente a los usuarios, con lo que los beneficios son mayores.

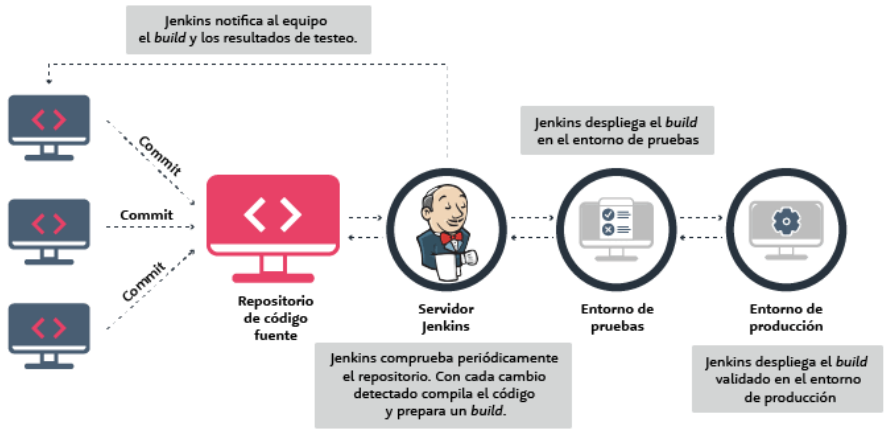
Qué se puede hacer

- Con Jenkins podemos automatizar multitud de tareas que nos ayudarán a reducir el time to market de nuestros productos digitales o de nuevas versiones de ellos. Concretamente, con esta herramienta podemos:
  - Automatizar la compilación y testeo de software.
  - Notificar a los equipos correspondientes la detección de errores.
  - Desplegar los cambios en el código que hayan sido validados.
  - Hacer un seguimiento de la calidad del código y de la cobertura de las pruebas.
  - Generar la documentación de un proyecto.
  - Podemos ampliar las funcionalidades de Jenkins a través de múltiples plugins creados por la comunidad, diseñados para ayudarnos en centenares de tareas, a lo largo de las diferentes etapas del proceso de desarrollo.

Cómo funciona

- Para entender cómo funciona Jenkins vamos a ver un ejemplo de cómo sería el flujo de integración continua utilizando esta herramienta:
  1. Un desarrollador hace un commit de código en el repositorio del código fuente.
  2. El servidor de Jenkins hace comprobaciones periódicas para detectar cambios en el repositorio.
  3. Poco después del commit, Jenkins detecta los cambios que se han producido en el código fuente. Compila el código y prepara un build. Si el build falla, envía una notificación al equipo en cuestión. Si resulta exitoso, lo despliega en el servidor de testeo.
  4. Después de la prueba, Jenkins genera un feedback y notifica al equipo el build y los resultados del testeo.
  5. Jenkins continúa revisando el repositorio frecuentemente y todo el proceso se repite.

Cómo funciona



Pros y Contras

- Ventajas
  - Es sencilla de instalar.
  - Es una herramienta opensource respaldada por una gran comunidad.
  - Es gratuita.
  - Es muy versátil, gracias a sus centenares de plugins.
  - Está desarrollada en Java, por lo que funciona en las principales plataformas.
- Desventajas
  - Su interfaz de usuario es anticuada y poco intuitiva, aunque puede mejorarse con plugins como Blue Ocean.
  - Sus pipelines son complejas y pueden requerir mucho tiempo de dedicación a las mismas.
  - Algunos de sus plugins están desfasados.
  - Necesita de un servidor de alojamiento, que puede conllevar configuraciones tediosas y requerir ciertos conocimientos técnicos.
  - Necesita ampliar su documentación en algunas áreas.

Pipeline

- Un pipeline es una forma de trabajar en el mundo DevOps, bajo la Integración Continua, que nos permite definir el ciclo de vida completo de un desarrollo de software.
- Un pipeline consiste en un flujo comprendido en varias fases que van en forma secuencial, siendo la entrada de cada una la salida de la anterior. Es un conjunto de instrucciones del proceso que sigue una aplicación desde el repositorio de control de versiones hasta que llega a los usuarios.
- El pipeline estaría formado por un conjunto de procesos o herramientas automatizadas que permiten que los desarrolladores como otros roles, trabajen de forma coherente para crear e implementar código en un entorno de producción.
- Cada cambio en el software, lleva a un complejo proceso hasta que es desplegado. Este proceso incluye desde construir software de forma fiable y repetible (conocido como “build”), hasta realizar todos los pasos de testing y los despliegues necesarios.
- Un pipeline Jenkins es un conjunto de plugins que soporta la implementación e integración de pipelines (tuberías) de despliegue continuo en Jenkins.
- Un pipeline Jenkins provee un gran conjunto de herramientas para dar forma con código a un flujo de entrega de la aplicación.

Definición

- La definición de un pipeline Jenkins, se escribe en un fichero de texto (llamado Jenkinsfile) que se puede subir al repositorio junto con el resto del proyecto de software.
- Ésta es la base del “Pipeline como código”: tratar el pipeline de despliegue continua como parte de la aplicación para que sea versionado y revisado como cualquier otra parte del código.
- La creación de un Jenkinsfile y su subida al repositorio de control de versiones, otorga una serie de inmediatos beneficios:
  - Crear automáticamente un pipeline de todo el proceso de construcción para todas las ramas y todos los pull request.
  - Revisión del código en el ciclo del pipeline.
  - Auditar todos los pasos a seguir en el pipeline
  - Una sencilla y fiable fuente única, que puede ser vista y editada por varios miembros del proyecto.

Sintaxis Básica Declarativa

https://www.jenkins.io/doc/book/pipeline/syntax/

- Pipeline {} Identificamos dónde empieza y termina el pipeline así como los pasos que tiene
- Agent. Especificamos cuando se ejecuta el pipeline. Uno de los comandos más utilizados es any, para ejecutar el pipeline siempre y cuando haya un ejecutor libre en Jenkins.
- Node (nodo): Máquina que es parte del entorno de Jenkins y es capaz de ejecutar un Pipeline Jenkins. Los nodos son agrupaciones de tareas o steps que comparten un workspace.
- Stages (etapas). Bloque donde se definen una serie de etapas a realizar dentro del pipeline.
- Stage. Son las etapas lógicas en las que se dividen los flujos de trabajo de Jenkins. Bloque que define una serie de tareas realizadas dentro del pipeline, por ejemplo: build, test, deploy, etc. Podemos utilizar varios plugins en Jenkins para visualizar el estado o el progreso de estos estados.
- Steps (pasos). Son todos los pasos a realizar dentro de un stage. Podemos definir uno o varios pasos.
- Step. Son las etapas lógicas en las que se dividen los flujos de trabajo de Jenkins. Es una tarea simple dentro del pipeline. Fundamentalmente es un paso donde se le dice a Jenkins qué hacer en un momento específico o paso del proceso. Por ejemplo, para ejecutar un comando en shell podemos tener un paso en el que tengamos la línea sh ls para mostrar el listado de ficheros de una carpeta.

Jenkinsfile

```
pipeline {
   agent any
   triggers { // Sondear repositorio a intervalos regulares
      pollSCM('* * * * *')
   }
   stages {
      stage ('Initialize') {
         steps {
            sh '''
               echo "PATH = ${PATH}"
               echo "JAVA_HOME = ${JAVA_HOME}"
               echo "JENKINS_VERSION = ${JENKINS_VERSION}"
            '''
         }
      }
```


Jenkinsfile

```
   stage("Compile") {
      steps {
         sh "mvn compile"
      }
   }
   stage("Unit test") {
      steps {
         sh "mvn test"
      }
   }
   stage("SonarQube Analysis") {
      steps {
         withSonarQubeEnv('SonarQubeDockerServer') {
            sh 'mvn clean verify sonar:sonar'
         }
      }
   }
```

Jenkinsfile

```
        stage("Build") {
            steps {
                sh "mvn package"
            }
        }
        stage("Deploy") {
            steps {
                sh "mvn install"
            }
        }
    }
}
```