P09- Pruebas de aceptación: JMeter

Pruebas de aceptación de aplicaciones Web

El objetivo de esta práctica es automatizar pruebas de aceptación de pruebas emergentes NO funcionales sobre una aplicación Web. Utilizaremos la herramienta de escritorio **JMeter**.

Usaremos una aplicación web: JPetStore. Implementaremos un plan de pruebas, en el que proporcionaremos la secuencia de entradas que nos permitirán evaluar si nuestro sistema soporta una determinada carga de usuarios (pruebas de carga)

GitHub

El trabajo de esta sesión también debes subirlo a *GitHub*. Todo el trabajo de esta práctica deberá estar en el directorio **P09-JMeter** dentro de tu espacio de trabajo.

Aplicación web jpetstore. Despliegue en Wildfly

Utilizaremos la aplicación Web Java JPetStore (mybatis.github.io/spring/sample.html) sobre la que haremos las pruebas con JMeter

En el directorio de plantillas proporcionamos el war de dicha aplicacón: ***jpetstore.war***

Ahora vamos a desplegar el war usando la consola de administración de Wildfly.

> **NOTA**: En una práctica anterior hemos descargado el servidior de aplicaciones Wildfly (versión 35.0.1), la cual usa las nuevas versiones de servlets de Jakarta. La aplicación jpetstore actualmente todavía no ha migrado a las librerías de Jakarta por lo no podremos usar Wildfly 35. Necesitamos usar la versión 26.1.3 (ver enlace de descarga).
>
> Simplemente hay que descomprimir el fichero descargado, por ejemplo en nuestro $HOME, y se creará el directorio `wildfly-26.1.3.Final`.

Primero vamos a **crear un usuario** con permisos de administrador para poder acceder a la consola de administración de Wildfly. Suponiendo que hemos instalado el servidor en nuestro $HOME, nos situamos en la carpeta `wildfly-26.1.3.Final/bin` y ejecutamos el comando:

```
> ./add-user.sh
```

Al ejecutar el *script* tendremos que introducir los **detalles de la cuenta**:

- Elegimos la opción `a) Update the existing user password and roles`
- A continuación introducimos el login y password, por ejemplo: login: `admin`, password: `adminadmin` (nos avisará de que no es una contraseña segura pero omitimos la advertencia y confirmamos que la contraseña es correcta y la introducimos de nuevo).
- Cuando nos pregunte `What groups do you want this user to belong to?`, simplemente pulsamos *enter*.
- Cuando nos pregunte `Is this new user going to be used for one AS process...?` contestamos que `no`.

Una vez creado el usuario, ponemos en marcha el servidor, usando el comando;

```
> ./standalone.sh
```

En el terminal veremos los ***logs*** del proceso, y al final, veremos el mensaje:

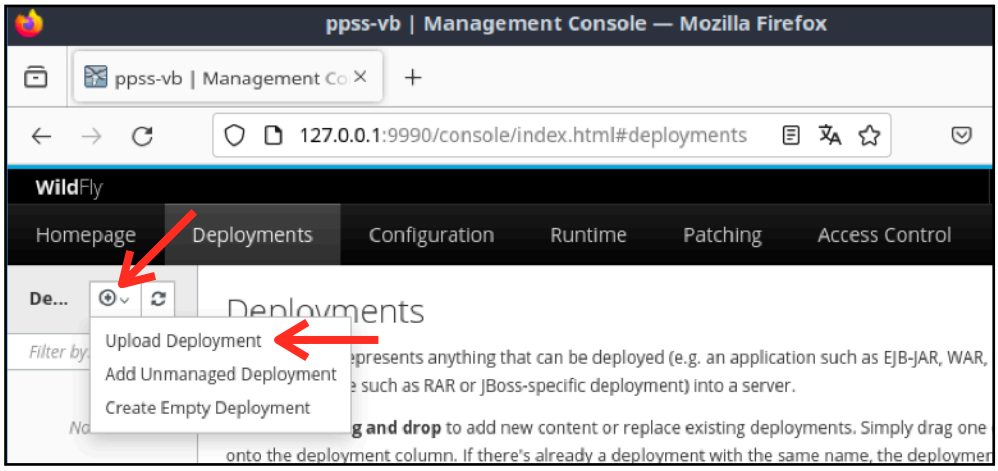
```
...
[org.jboss.as] (Controller Boot Thread) WFLYSRV0051: Admin console listening on
http://127.0.0.1:9990
```

Para acceder a la consola de administración de Wilfly, usamos la dirección `http://127.0.0.1:9990` en el navegador.

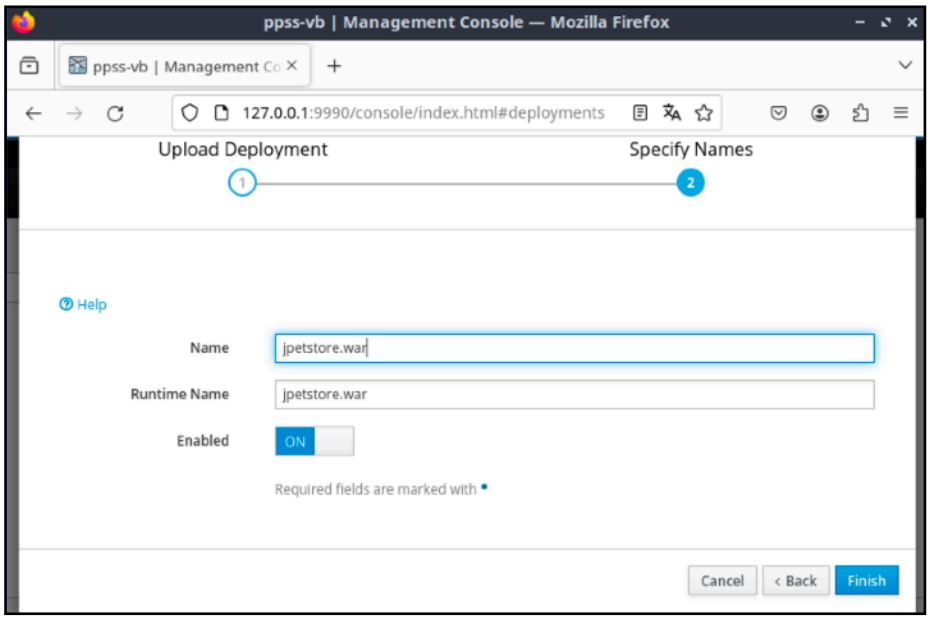
Veremos un mensaje advirtiéndonos de que estamos usando una versión antigua. Simplemento lo ignoramos y cerramos la ventana emergente.

Accedemos a la pestaña `Deployments` y desde el icono `+` seleccionamos la opción `Upload Deployment` (**Figura 1**).

Figura 1. Consola de administración de Wildfly. Despliegamos el war.



Una vez seleccionado el fichero ***war*** a desplegar, en el siguiente paso (paso 2) indicamos `jpetstore.war` en los campos ***Name*** y ***Runtime name*** y con esto finalizamos el proceso de despliegue de nuestra aplicación web. (**Figura 2**).



Ahora ya podemos acceder a la aplicación desde el navegador usando la url:

```
http://localhost:8080/jpetstore
```

A partir de ahora, para acceder a nuestra aplicación web sólo tendremos que arrancar el servidor (usando el ***script*** `standalone.sh` desde el terminal) y acceder a la aplicación desde el navegador, con:

```
http://localhost:8080/jpetstore
```

Es importante NO tener abierto IntelliJ ni ningún otro proceso adicional que no sea absolutamente necesario, ya que interferirán en nuestro proceso de pruebas, desvirtuando el resultado obtenido.

IMPORTANTE: Necesitaremos usar un proxy para poder grabar las acciones del navegador desde JMeter. En la máquina virtual no nos ha sido posible configurar el proxy para usarlo con Chrome (no se puede acceder a la configuración del sistema desde las opciones de Chrome. Modificando los parámetros desde el terminal, aunque el navegador si parece que está usando el proxy y podemos ejecutar la aplicación web, aparecen errores en el log del terminal y realmente el proxy de jmeter no graba las acciones del navegador). Por lo que para esta sesión usaremos **Firefox** en lugar de Chrome.

Ejercicios

Si usas la máquina virtual, para iniciar JMeter ejecuta desde el terminal:

```
> jmeter
```

Nota1: Cuando arrancamos JMeter, por defecto la apariencia de la aplicación es en modo "*Darcula*", con lo cual tendrá un aspecto bastaste oscuro. Si preferís cambiarlo a otro más claro, podéis hacerlo desde *Options→Look and Feel*, y seleccionar por ejemplo "*Intellij*".

Nota2: En la máquina virtual hemos instalado jmeter 5.6.3

Si no usas la máquina virtual tendrás que descargarlo desde su página oficial: (https://jmeter.apache.org/download_jmeter.cgi).

➪ ➪ Ejercicio 1: Uso de Proxies en JMeter

Antes de comenzar a implementar nuestros tests, vamos a explicar el **uso de "proxies"** en JMeter para averiguar los parámetros de una petición al servidor cuando éstos no son visibles en la url correspondiente.

La creación de un plan de pruebas con JMeter puede presentar cierta dificultad cuando se ven implicadas *queries* y/o formularios complejos, peticiones https POSTs, así como peticiones javascript, en las que los parámetros que se envían por la red NO son visibles en la URL.

JMeter proporciona un **servidor HTTP *proxy***, a través del cual podemos utilizar el navegador para realizar las pruebas, y JMeter "grabará" las peticiones http generadas, tal y como se enviarán al servidor, y creará los correspondientes HTTP *samplers*. Una vez que hayamos "guardado" las peticiones HTTP que necesitemos, podemos utilizarlas para crear un plan de pruebas.

Vamos a ver paso a paso cómo utilizar dicho servidor *proxy*:

A) JMeter "grabará" todas las acciones que realicemos en el navegador en un controlador de tipo "recording", por lo que tendremos que incluir un controlador de este tipo en nuestro plan de pruebas. Lo primero que haremos será **añadir un grupo de hilos** en nuestro plan. Desde el menú contextual del nodo "Test Plan", elegiremos *Add→Threads (Users)→**Thread Group.*** Asegúrate de que está configurado con 1 hilo, un periodo de subida de 1 segundo, y 1 iteración. A continuación, desde el menú contextual del grupo de hilos que acabamos de añadir tendremos que seleccionar *Add→Logic Controller→ **Recording Controller***.

B) Ahora vamos a **añadir un servidor *proxy*** para realizar peticiones HTTP. Desde el menú contextual del elemento *Test Plan*, seleccionando "*Add→Non-Test Elements→**HTTP(S) Test Script Recorder***". En la CONFIGURACIÓN del *proxy* tendremos que asegurarnos de que:

- el **puerto** no está ocupado con algún otro servicio en nuestra máquina local, (ver el valor del campo *Global Settings→Port* de la configuración del elemento *HTTP(S) Test Script Recorder*, por defecto tiene el valor 8888). En nuestro caso, los puertos 8080 y 9990 están ocupados por el servidor de aplicaciones Wildfly, por lo que en principio el puerto 8888 estará libre
- el campo ***Target Controler*** (en la pestaña de configuración *Test Plan Creation*) tiene el valor ***Test Plan >Thread Group >Use Recording Controller***, de esta forma, nos aseguramos de que los samplers se añadirán al controlador de grabación que acabamos de añadir a nuestro plan..

Cómo averiguar qué puertos están siendo utilizados

Para averiguar qué conexiones están activas y qué procesos y puertos están abiertos podemos usar el comando:

> `ss -tap` (https://www.binarytides.com/linux-ss-command/)

De forma alternativa, también puedes usar el comando:

> `lsof -i tcp`

Si queremos comprobar que el puerto 8888 no está siendo usado por ningún proceso podemos utilizar el comando: `lsof -i:8888` (si el puerto 8888 está libre no veremos ningún resultado. Si el puerto 8888 está siendo usado el comando nos mostrará qué proceso lo está usando).

Por defecto, el servidor proxy (elemento *HTTP(S) Test Script Recorder*) interceptará todas las peticiones http dirigidas a nuestra aplicación y las "grabará" en el controlador de tipo ***recording*** que hemos añadido a nuestro plan de pruebas. Por defecto, JMeter grabará cualquier "cosa" que envíe o reciba el navegador, incluyendo páginas HTML, ficheros javascript, hojas de estilo css, imágenes, etc, la mayoría de las cuales no nos serán de mucha utilidad para nuestro plan de pruebas. Para "filtrar" el tipo de contenido que queremos (o no queremos) utilizaremos patrones URL a Incluir (o a Excluir), desde la pestaña ***Request Filtering*** de la configuración del Servidor Proxy.

Por ejemplo, vamos a **excluir** del proceso de grabación ciertas **imágenes** y **hojas de estilo**, entre otros (desde la pestaña ***Request Filtering***, y más concretamente desde el elemento ***URL Patterns to exclude***). Para ello tendremos que pulsar con el ratón sobre el botón *Add*, y a continuación hacer doble click en la nueva fila añadida (en color blanco). Tenemos que añadir cuatro líneas para escribir las siguientes cuatro expresiones regulares:

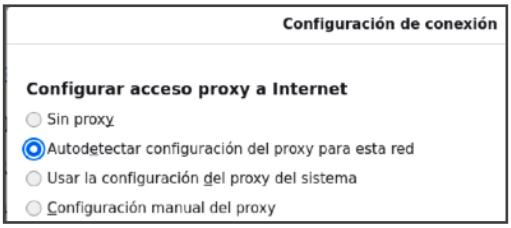
```
.*\.gif
.*\.css
.*\.txt
.*(socket.io).*
```

que representan los patrones URL a excluir (cada patrón irá en una línea diferente). NO hay que poner comillas al introducir cada patrón. Tendremos que hacer doble click con el botón izquierdo del ratón en cada una de las líneas para poder escribir los valores en ellas. Para más información sobre expresiones regulares podéis consultar: http://rubular.com).

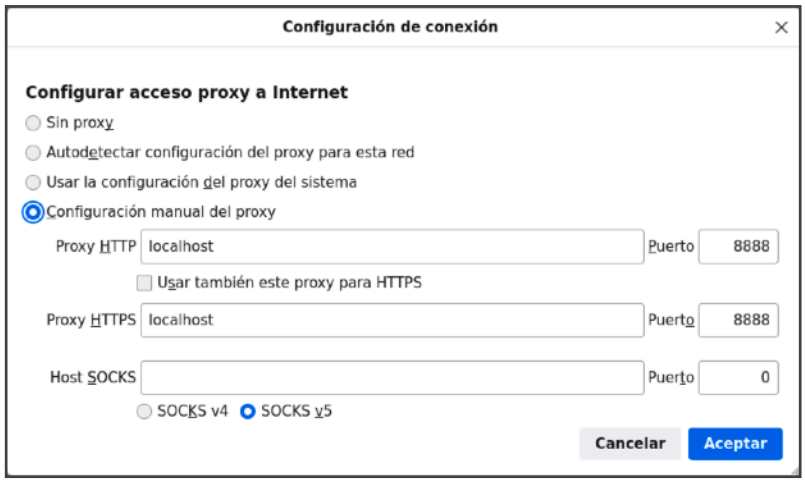
De esta forma evitamos grabar peticiones que no vamos a necesitar, por ejemplo ignoraremos las peticiones de ficheros de imagen de tipo .gif, ficheros de estilo, ficheros de texto y peticiones generadas por el api websocket. Puedes probar a quitar los patrones a excluir y verás que se graban muchas más peticiones de las que necesitamos. No pasa nada porque podemos borrar posteriormente cualquiera de ellas.

C) A continuación tendremos que **configurar el navegador** (en este caso Firefox) para utilizar el puerto en donde actuará el proxy JMeter. Para ello iremos al menú de Ajustes→ General→Configuración de red→Configuración... (al final de la página) "Preferencias→ General→Network settings→Settings...". Desmarcamos la opción actual ("Use system proxy settings"), y marcamos "Manual proxy configuration". Como valor de Proxy HTTTP pondremos ***localhost***, y el puerto el **8888**. Acuérdate de marcar la casilla "Also use this proxy for FTP and HTTPS".

Las siguientes capturas de pantalla muestran la configuración Inicial de Firefox, y la configuración que debemos usar para utilizar el proxy JMeter.



Configuración Inicial de Firefox



Configuración de Firefox para usar el proxy de JMeter

Finalmente, y dado que las conexiones con localhost o 127.0.0.1 nunca se establecen a través de un proxy, tendremos que cambiar el valor de la variable ***network.proxy.allow_hijacking_localhost*** a `true`, desde about:config

Valor de la propiedad `network.proxy.allow_hijacking_localhost` para usar el proxy de JMeter

Ahora ya estamos en disposición de usar el *proxy* de JMeter para ver cuáles son las peticiones http que tenemos que utilizar para "loguearnos" en el sistema. La petición de *login* es una petición http de tipo POST, y los parámetros de esta petición NO son visibles en el navegador, por lo que averiguaremos dichos valores a través del *proxy*.

D) Una forma de asegurarnos de que el navegador está utilizando el proxy de JMeter es: estando inactivo el proxy intentar navegar desde Firefox (o acceder a nuestra aplicación web). Si el navegador sigue sirviendo las páginas es que NO está utilizando el proxy.

Una vez que nos hemos asegurado de que hemos configurado adecuadamente Firefox para utilizar el proxy de JMeter, iremos al panel de configuración de nuestro *Test Script Recorder* y en el panel "*State*" veremos el botón "*Start*". Al pulsar dicho botón **pondremos en funcionamiento el proxy** y podemos comenzar la grabación (no podremos arrancar el proxy si previamente no hemos añadido el controlador de grabación en nuestro plan).

E) Al inciar el proxy (pulsando sobre el icono "Start" desde el panel del configuración correspondiente) nos aparecerá una alerta indicando que se ha creado un certificado temporal en un directorio de JMeter (simplemente tenemos que pulsar sobre "OK" en dicha ventana). También veremos una ventana con el nombre "*Recorder: Transactions Control*" que permanecerá abierta hasta que detengamos el proxy. Ahora, cualquier acción que hagamos desde el navegador, quedará "registrada" en el controlador de grabación que hemos añadido en nuestro plan de pruebas. Vamos a realizar las siguientes acciones en el navegador:

- Entramos en la aplicación de la tienda de animales (`http://localhost:8080/jpetstore/actions/Catalog.action`), a continuación pinchamos sobre ***Sign In***.
- Introducimos las credenciales del usuario: (login) **j2ee**, (contraseña) **j2ee**. Se trata de un usuario que se crea por defecto en la aplicación, y pulsamos sobre "Login".
- Finalmente pulsamos sobre ***Sign Out*** en nuestra aplicación web y PARAMOS la grabación desde el proxy de JMeter (o bien pulsando en el botón con el círculo rojo de la ventana "*Recorder: Transaction Control*").

Nota: también podriamos haber creado previamente un nuevo usuario, o incluso durante la grabación. No hay problema porque aunque grabemos acciones que no necesitamos, podremos borrarlas en cualquier momento.

Ahora podemos ver las peticiones http que han quedado grabadas en el "Controlador Grabación" de nuestro plan. Selecciona cada una de ellas y fíjate en la ruta y parámetros de petición (**comprueba que cuando hacemos "login" en el sistema se envían 5 parámetros, y que es una petición POST**). Utilizaremos esta información en el siguiente ejercicio para confeccionar nuestro plan de pruebas.

Observarás que hay algunas peticioness que están deshabilitadas (aparecen en gris claro). Se trata de redirecciones realizadas por la aplicación, que también quedan grabadas, pero que no se tendrán en cuenta cuando ejecutemos el test.

En cualquier momento podemos "habilitar/deshabilitar" cualquier *sampler* usando su menú contextual con la opción **Disable/Enable**. La opción ***Toogle*** cambia el estado del *sampler* (si está deshabilitado lo habilita y viceversa).

De los *samplers* (peticiones http) que hemos "grabado", nos interesan las siguientes:

- la petición **GET** para acceder al catálogo, con la URL *http://localhost:8080/jpetstore/actions/Catalog.action*
- la petición **GET** con el parámetro "***signonForm***", que es la que solicita al servidor el formulario para rellenar las credenciales del usuario. (Si te fijas, cuando en la página seleccionamos "Sign In", verás en el navegador la ruta (localhost:8080/jpetstore/actions/Account.action?signonForm=)
- la petición **POST** que usamos para introducir las credenciales. Verás que la petición http que generamos tiene 5 parámetros, pero, a diferencia de las peticiones GET, en una petición POST los valores de los parámetros NO se muestran en la URL del mensaje HTTP.
- la petición **GET** con el parámetro "***signoff***" que usamos para "salir" de nuestra cuenta cuando en la página seleccionamos "*Sign Out*"

Revísa las peticiones grabadas y habilita sólo esas cuatro. Guarda el plan de pruebas que hemos creado, lo puedes hacer desde "*File→Save*", o pulsando sobre el icono con forma de *disquette* y ponle el nombre **Plan-Ejercicio1-proxy.jmx**

Una vez terminado el ejercicio recuerda volver a la configuración inicial de red en Firefox para poder acceder a Internet sin utilizar el proxy de JMeter. Además tendrás que volver a poner a false el valor de la variable network.proxy.allow_hijacking_localhost desde about:config.

F) Vamos a ejecutar nuestro plan pulsando el botón con el icono con forma de triángulo verde. (**Nota**: asegúrate de que solamente están habilitados los cuatro *samplers* que hemos indicado en la apartado anterior (con sus elementos hijo). Si has grabado más acciones de las necesarias, deshabilítalas desde el menú contextual del elemento seleccionando (*Disable*)

Observaciones sobre el CONTROLADOR DE GRABACIÓN: Cuando ejecutemos el plan de pruebas, el "*Recording Controller*" no tiene ningún efecto sobre la lógica de ejecución de las acciones que contiene, en este sentido es como el controlador "Simple", es decir, actúa como un nodo que "agrupa" un conjunto de elementos.

¿Cómo sabes si se ha ejecutado o no el plan? Durante la ejecución del mismo puedes ver en la parte superior derecha de la aplicación el número de hilo en ejecución y el número total de hilos a ejecutar. También nos muestra el número de errores que se han producido durante la ejecución. Además, durante la ejecución, el círculo gris cambiara a color verde.

Dado que nuestro plan está formado por 4 peticiones http, y solamente tenemos un usuario, ni siquiera veremos "incrementarse" los números de los hilos de ejecución que estamos ejecutando, de hecho el temporizador es bastante probable que siga a cero, ya que la ejecución del plan consumirá un tiempo inferior a un segundo (y por supuesto no apreciaremos que el círculo se muestra en color verde en ningún momento, todo dependerá del ordenador que estés usando).

No podemos saber nada sobre la ejecución del plan, ya que NO HEMOS "capturado" los datos de la ejecución en ningún LISTENER!

Vamos a añadir un listener a nuestro plan de pruebas (desde el menú contextual del "Test Plan"). En concreto añadiremos el listener "***View Results Tree***", el cual nos mostrará las peticiones y respuestas de cada uno de los *samplers* del plan a medida que se van ejecutando.

Una vez que has añadido el listener, sitúate con el ratón en el elemento Test Plan o en el Grupo de Hilos. Vuelve a ejecutar el plan.

Volverás a tener la "sensación" de que no ha ocurrido nada. Sin embargo, si ahora nos vamos al nodo que contiene el listener y podemos ver TODOS los mensajes http (tanto de petición como de respuesta) de nuestro plan.

Si ejecutas el plan y estás "posicionado" sobre el listener verás como se muestra la información a medida que se va ejecutando dicho plan.

Otra cosa que debes observar es que si ejecutamos 3 veces el plan, en el listener quedan registradas las 3 ejecuciones (los datos no se borran de forma automática entre diferentes ejecuciones).

Para "limpiar" los datos almacenados en los listeners podemos usar los iconos con forma de "escoba" de la barra de herramientas.

Este listener consume bastante memoria de forma que afectará a los datos recopilados por jmeter. Ya hemos explicado en clase que en una prueba real, no usaremos este listener ni ningún otro que muestre los datos de forma gráfica, ya que interferirán en los resultados obtenidos.

A efectos de comprobar lo que hemos comentado en el apartado anterior sobre la información mostrada en la parte superior derecha de la barra de herramientas, cambia la configuración del grupo de hilos de forma que tengamos 50 hilos y 2 iteraciones. Vuelve a ejecutar el plan. Es probable que ahora sí veas el círculo verde y que veas el contador de los hilos en ejecución como va cambiando durante la ejecución. Si sigues sin apreciar nada incrementa el número de hilos y/o las iteraciones hasta que consigas ver los cambios.

Si cada vez que ejecutas el plan estás situado sobre el listener, verás que se irá incrementando considerablemente la cantidad de datos que muestra, a menos que vayamos "limpiando" los resultados entre ejecuciones consecutivas.

Finalmente vuelve a guardar el plan de pruebas.

➪ ➪ Ejercicio 2:Driver y validación de carga

Vamos a crear un nuevo plan de pruebas en el que usaremos la información obtenida en el ejercicio anterior. Puedes duplicar el fichero *jmx* del ejercicio anterior, y ponerle como nombre **Plan-Ejercicio2.** Queremos evaluar el rendimiento de nuestra aplicación y comprobar si es capaz de soportar una determinada carga de usuarios.

Llamaremos a nuestro plan de pruebas "**Plan JMeter**". De momento vamos a mantener nuestro **grupo de hilos** con los valores por defecto: un hilo, un periodo de subida de 1 segundo y un bucle con una única iteración. Borramos el elemento *HTTP(S) Test Script Recorder* puesto que ya no nos hace falta. A continuación añadiremos los siguientes elementos:

A) Primero vamos a incluir dos **elementos de configuración** a nuestro grupo de hilos. Los elementos de configuración evitan que tengamos que introducir repetidamente alguna información de configuración de los *samplers*. Por ejemplo, si en el plan de pruebas incluimos varias llamadas a algunas páginas que están protegidas mediante una autenticación http básica, se podría compartir la información de usuario y password, para no tener que repetir estos datos para cada *sampler* dentro del plan de pruebas. Para pruebas de aplicaciones Web , el elemento de configuración más importante es "**HTTP Request Defaults**" (ver http_request_defaults component). Lo añadimos desde el menú contextual del grupo de hilos (*Add→ Config Element→HTTP Request Defaults*). Estableceremos como nombre del servidor ***localhost***, el número de puerto será ***8080***

Otro elemento de configuración muy útil es "**HTTP Cookie Manager**". Éste almacena cookies y hace que estén disponibles para subsecuentes llamadas al mismo sitio, tal y como haría un navegador.

Dado que cada hilo representa un usuario diferente, las cookies no se compartirán entre hilos. Lo añadiremos desde el menú contextual del grupo de hilos (*Add→ Config Element→HTTP Cookie Manager*). En este caso usaremos la configuración por defecto de este elemento.

Si nos equivocamos al crear un elemento, en cualquier momento podemos borrar/modificar/mover cualquier elemento del plan. Para **borrar** un elemento lo haremos desde el menú contextual de dicho elemento. Para **modificar** su configuración simplemente seleccionaremos dicho elemento. Para **mover** un elemento a otra posición dentro del plan, seleccionaremos y arrastraremos dicho elemento a su nueva posición.

B) Ahora vamos a añadir un ***sampler*** (a nuestro grupo de hilos) para hacer peticiones de páginas ***http*** a nuestro servidor de aplicaciones y así simular la acción de los usuarios (desde el menú contextual del grupo de hilos: ***Add→Sampler→HTTP Request***). La URL inicial de la tienda es "http://localhost:8080/jpetstore". Se trata de una petición GET. El servidor y el puerto ya los hemos indicado en un elemento de configuración (localhost y 8080), por lo que no es necesario indicarlos. Después del contexto inicial (cuyo valor es *jpetstore,* y es donde se encuentra desplegada nuestra aplicación web), vendría el nombre del recurso, en este caso, la página html a la que queremos acceder. Observamos que no aparece esta información (no es necesaria ya que por defecto se accederá a index.html). No será necesario indicar nada en el campo Path, ya que es el valor que hemos registrado en el elemento de configuración HTTP Request Defaults. Como vamos a usar varios *samplers* http, será conveniente indicar un nombre adecuado para poder diferenciarlos y hacer más "legible" nuestro código. Por ejemplo, para este primer *sampler* podemos especificar como nombre el valor "**Home**"

C) Para asegurarnos de que estamos en la página correcta, vamos a añadir una **aserción** (desde el menú contextual del elemento Home: ***Add→Assertions→Response Assertion***). Nos aseguraremos de marcar "***Text Response***", y el patrón a probar contendrá la cadena "*Welcome to JPetStore 6*". Recuerda que primero tienes que pinchar sobre el botón "***Add***", y después hacer doble click en la fila en blanco añadida para editar su contenido. El texto se pone SIN comillas. Otra opción es copiar el texto anterior (con ctrl-C) y directamente seleccionar el botón ***Add From Clipboard***. Llamaremos a este elemento ***Assert-Welcome***

D) Ahora añadiremos **otra petición http** GET (al grupo de hilos). En este caso usaremos el sampler que hemos grabado con la petición GET a la url: http://localhost:8080/jpetstore/actions/Catalog.action. Puedes moverlo desde el elemento Recording Controller y situarlo después de Home (al mover este elemento, también movemos todos sus "desdendientes", en este caso el elemento HTTP Header Manager). Ya que esta petición nos muestra el catálogo de animales, podemos asignarle el nombre "**Main Catalog**". Modifica la configuración del sampler para eliminar redundancias con el elemento de configuración correspondiente. Seguidamente anidamos en el sampler una aserción de respuesta, en la que verificaremos el texto "Saltwater". El nombre del elemento será ***Assert-Saltwater***.

E) Queremos acceder al menú de Reptiles (pinchando sobre la imagen correspondiente). Añadimos otra petición http (a la que ponemos como nombre **Reptiles.** Para averiguar la url de nuestra petición puedes hacerlo desde el navegador: entra en la tienda y "pincha" sobre la imagen de reptiles. Podemos comprobar que la URL del navegador, es la siguiente: "http://localhost:8080/jpetstore/actions/Catalog.action?viewCategory=&categoryId=REPTILES" En este caso, vemos que la petición tiene dos parámetros (los parámetros se especifican con pares nombre=valor, separados por '&'): un primer parámetro con nombre ***viewCategory*** (que en este caso no tiene valor asociado, pero que es necesario incluir), y un segundo parámetro con nombre ***categoryId*** (cuyo valor es **REPTILES**). Configura el *sampler* teniendo en cuenta esta información (el path y los parámetros de la petición). Añade una **aserción de respuesta** con el texto "Reptiles" (ponle de nombre ***Assert-Reptiles***). Observa que en la configuración del elemento Response Assertion aparece al final: "*Custom Failure Message*". Puedes incluir mensajes en todas la aserciones, los cuales se mostrarán si la aserción falla.

Nota: Cuando utilizamos ***samplers* http**, para averiguar la "ruta" y parámetros de la petición URL podemos fijarnos en la ruta que aparece en el navegador al ejecutar la aplicación. Esta aproximación funciona bien cuando se trata de peticiones GET "simples" y parámetros "fáciles de manipular". Páginas que contengan formularios grandes pueden requerir docenas de parámetros. Además, si se utilizan peticiones HTTP POST (como por ejemplo el envío de login y password), los valores de los parámetros no aparecerán en la URL, por lo que tendremos que descubrirlos de alguna otra forma. Por ejemplo usando el **Proxy JMeter**, que grabará el caso de prueba por nosotros, tal y como hemos visto en el ejercicio 1, y que luego podremos modificar según nuestras necesidades.

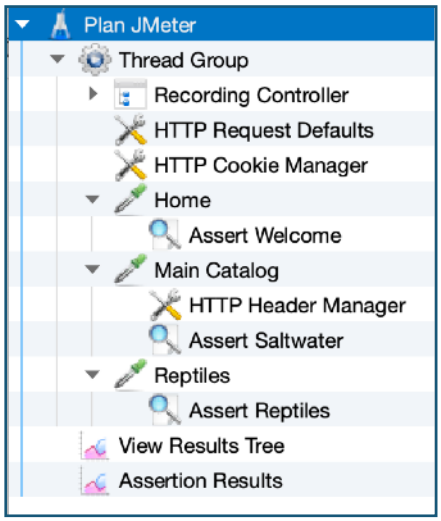
F) Antes de continuar vamos a **grabar nuestro plan** de pruebas (como medida de precaución es recomendable grabar a menudo). Lo podemos hacer desde "*File→Save*". Dado que podemos ejecutar el plan en cualquier momento, vamos a hacerlo ahora, pero antes nos aseguraremos de que en las propiedades del grupo de hilos solamente se genera un hilo, con un periodo de subida de 1 segundo y una única vez. Pulsamos el icono con un triángulo verde (el primero de ellos).

Para poder "ver" lo que ha ocurrido al ejecutar el plan usa el listener ***View Results Tree.*** Si ejecutas el plan estando situado sobre el listener, verás cómo se actualiza en tiempo real. Recuerda que los resultados se van a ir "acumulando" en el listener si no los "limpias" entre dos ejecuciones sucesivas!!

Vamos a añadir un segundo listener a nuestro plan de pruebas con ***Add→Listener→Assertion Results***, el cual nos muestra los nombres de cada sampler y los errores producidos.

Vuelve a **ejecutar el plan**. Puedes realizar varias ejecuciones para familiarizarte con la información que proporcionan estos dos listeners. (si todas las aserciones se evalúan a "true" verás que el listener correspondiente no muestra ninguna información. Prueba a cambiar la aserción para que se produzca un "fallo", y podrás observar que el listener muestra el error producido).

A la derecha mostramos el aspecto del plan de pruebas hasta este momento



Plan de pruebas hasta este momento

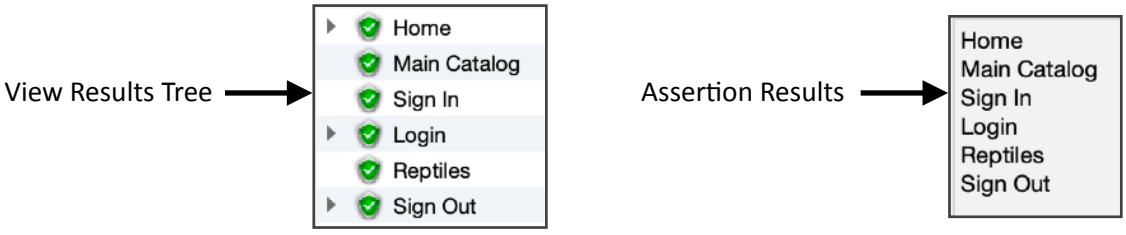
Antes de continuar, familiarízate con la información que proporciona cada uno de los listeners (mensajes http de petición y respuesta y resultados de las aserciones). Puedes probar a mover los listeners a otro nivel del plan (por ejemplo como hijos de algún sampler) y verás lo que ocurre. Observa también que la información mostrada por los listeners no se "resetea" entre dos ejecuciones consecutivas de forma automática.

G) Añadimos 2 samplers para acceder al catálogo de Perros y de Pájaros (pinchando en los correspondientes enlaces). Puedes duplicar el elemento "Reptiles" dos veces y modificar la configuración de las peticiones http y de las aserciones. Llamaremos a los nuevos samplers ***Dogs*** y ***Birds***, respectivamente. Y ***Assert-Dogs*** y ***Assert-Birds*** a las aserciones de respuesta asociada, en las que comprobaremos que en la página aparece el texto "Dogs" y "Birds".

H) Movemos (o copiamos) a continuación de Main Catalog los dos elementos que tenemos guardados en el controlador de grabación con la petición GET asociada a la selección de la opción *Sign In*, y a continuación la petición POST con las credenciales del usuario. Al primer sampler lo llamaremos ***Sign In***, y al segundo ***Login***. Al sampler *Sign In* le añadimos una aserción de respuesta con el patrón "Please enter your username and password" (y la llamaremos ***Assert-Enter Data***). Y al sampler *Login* le añadimos una aserción de respuesta que estará formada por dos patrones de texto "Sign Out", y el patrón "Welcome ABC!"(debes poner cada uno en una linea diferente. Recuerda que NO hay que poner las comillas). A la aserción de respuesta la llamaremos ***Assert-Welcome***

I) Movemos (o copiamos) después de *Birds*, el sampler de nuestro controlador de grabación correspondiente a seleccionar *Sign Out* (al que llamaremos ***Sign Out***). A dicho sampler le añadiremos una aserción de respuesta con el patrón "Sign In". (y la llamaremos Assert-Sign Out). Ahora ya podemos borrar el controlador de grabación.

Grabamos el plan hasta este momento. Ejecuta el plan de pruebas. Los listeners View Results Tree y Assertion Results deben mostrar lo siguiente:



J) Ahora usaremos un **controlador lógico** para "alterar" el comportamiento secuencial del plan. Añadimos un controlador ***Interleave*** desde el menú contextual del Grupo de Hilos: ***Add→Logic Controller→Interleave Controller***. Este controlador "alterna" la ejecución de uno de sus hijos en cada iteración de cada uno de los hilos. Podéis consultar el funcionamiento de cualquier componente JMeter en (http://jmeter.apache.org/usermanual/component_reference.html). Este controlador estará situado a continuación del salmpler Login. Si ejecutas de nuevo al plan, tal y como lo tenemos, debes ver los mismos resultados del apartado anterior en los dos listeners (ya que nuestro grupo de hilos solamente tiene 1 iteración). Prueba a configurar el grupo de hilos con dos iteraciones o más iteraciones y observa el resultado.

K) **Añadimos dos listeners más** (desde el nodo raíz de nuestro plan de pruebas): un ***Aggregate Report,*** que proporciona un resumen general de la prueba; y un ***Graph Results***, que "dibuja" los tiempos registrados para cada una de las muestras. El *Aggregate Report* muestra una tabla con una fila para cada petición, mostrando el tiempo de respuesta, el número de peticiones, ratio de error, ... El gráfico de resultados muestra el rendimiento de la aplicación (throughput) como el número de peticiones por minuto gestionadas por el servidor.

Vuelve a ejecutar el plan (usa una hilo y una iteración) y observa los resultados mostrados por los nuevos listeners que hemos añadido (Puedes comprobar de nuevo que los resultados obtenidos por los listeners no se "resetean" de forma automática). Reduce al máximo el número de aplicaciones "abiertas" en el ordenador cuando estés ejecutando JMeter, ya que afectarán a los datos obtenidos. En un caso de prueba "real" solamente deberíamos tener en ejecución la aplicación JMeter en una máquina.

L) Vamos a introducir pausas para simulara un comportamiento más realista. Para ello añadiremos **timers.** Añadimos un ***Uniform Random Timer*** desde el menú contextual del grupo de hilos. configuramos un ***Constant delay offset*** de 1500 ms, y un ***Random delay maximum*** de 100ms. Dado que el proceso de login le llevará al usuario algo más de tiempo, añadiremos otro timer del mismo tipo como hijo del sampler login. En este caso el *delay* aleatorio seguirá siendo de 100 ms máximo, mientras que el *offset* constante será de 2000ms. Para comprobar el efecto de los timers, añadiremos también el listener ***View Results in Table*** (desde nuesto plan), el cual nos proporciona, para cada muestra, el tiempo en el que comienza a ejecutarse.

Ejecuta de nuevo el plan (recuerda que primero debes "limpiar" los resultados de ejecuciones previas) y comprueba en el listener *View Results in Tabl*e que cada muestra se lanza teniendo en cuenta las pausas introducidas por los timers: el tiempo de comienzo de una muestra será tiempo de respuesta (*sample time*) de la muestra anterior + el delay impuesto por el timer. Observa que el tiempo de ejecución de la muestra no cambia al introducir los "delays" de los timers, en cambio el ***throughput*** sí se ve afectado por estos retrasos, ya que se calcula como: *numero_peticiones/ tiempo_total_ejecución*. El tiempo total de ejecución se calcula desde el inicio de la primera petición hasta el final de la última, y tiene en cuenta los tiempos de espera.

M) Ahora vamos a suponer que está prevista una carga de 25 usuarios. Nuestras especificaciones estipulan que con 25 usuarios concurrentes, el tiempo de respuesta para cada uno de ellos debe ser inferior o igual a 2ms segundos por página. Vamos a cambiar los parámetros del grupo de hilos de nuestro plan para validar si nuestra aplicación cumple con los requisitos de rendimiento acordados con el cliente.

Es importante que entiendas lo que significa el término "concurrentes": que los 25 usuarios estén siendo atendidos "a la vez". Por ejemplo, si lanzamos 100 usuarios con un ramp-up de 1 segundo, esto no garantiza que se vayan a ejecutar de forma concurrente. Supongamos que cada usuario

necesita 1 ms para ejecutar el plan. Después de un segundo, todos los usuarios se habrán ejecutado, pero no de forma concurrente, ya que cuando empieza el siguiente, el anterior ya habrá terminado.

Podemos comprobar el número de hilos que se están ejecutando de forma concurrente observando la gráfica "***Active threads Over Time***". Esta gráfica no la proporciona ningún listener. Vamos a generarla a través de un comando desde el terminal.

Para ver mejor los resultados, vamos a cambiar la "granularidad" del eje de abcisas de la gráfica. Por defecto tiene un valor de 1 minuto, nosotros usaremos sólo 3 segundos. Para ello tienes que editar el fichero `/home/ppss/apache-jmeter-5.5/bin/user.properties`, y modificar el valor de la propiedad `jmeter.report.generator.overall_granularity` (la encontrarás en la línea 76 del fichero, debes borrar el "#" para descomentarla):

```
jmeter.reportgenerator.overall_granularity=3000
```

A continuación añadimos el listener ***Simple Data Writer***, que guardará en un fichero los datos calculados por JMeter durante el ejecución del plan. Vamos a generar dicho fichero con el nombre ***data.csv*** en la misma carpeta que nuestro plan (tendrás que poner la ruta absoluta de dicho fichero en el cuadro de texto *Filename* de la configuración del listener).

Cambia el número de hilos del plan a 25, y usa, por ejemplo, un ramp-up de 50 segundos, con 1 iteración. Ejecuta el plan. Ahora sí podrás ver que el icono superior derecho cambia de color, y también cómo se va incrementando el contador que muestra el número de hilos en ejecución. Cuando termines la ejecución se habrá generado el fichero data.csv en tu carpeta de trabajo. Ahora generaremos los informes a partir de estos datos en la carpeta ***reports***.

Ejecuta el siguiente comando (desde la carpeta donde tienes el fichero data.csv):

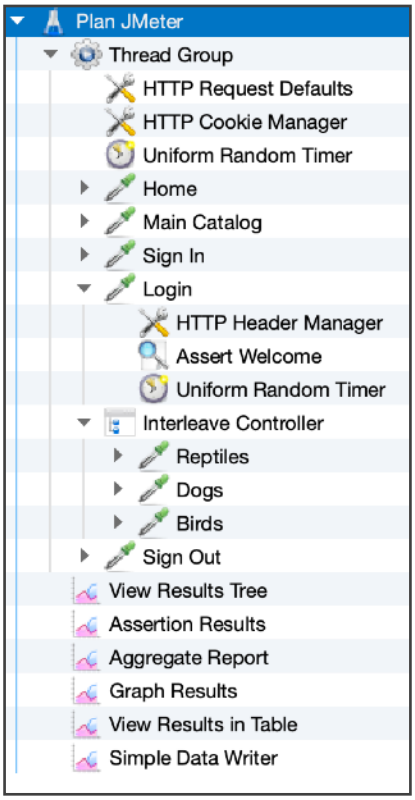
```
> jmeter -g data.csv -o reports/
```

En la carpeta ***reports***, abre el fichero *index.html* en el navegador. La gráfica que nos interesa es *Charts→Over Time→Active Threads Over Time*. Aquí verás la información del número de hilos activos cada tres segundos. Si en ningún momento el número de hilos concurrentes es inferior a 25 entonces tendrás que ajustar los parámetros ampliando el número de iteraciones (por ejemplo 5), y bajando el ramp-up, por ejemplo a 10 segundos. Si en la gráfica puedes ver los 25 hilos ejecutándose concurrentemente, entonces sólo tienes que comprobar en la tabla del Aggregate Report si los todos los tiempos de respuesta medio para cada sampler son inferiores a 2 ms.

Si volvemos a generar los informes mediante línea de comandos JMeter detectará que ya existe el fichero data.csv. Seleccionaremos la opción para sobreescribirlo. También tendremos que borrar el contenido del directorio "reports" porque no nos dejará sobreescribirlo. Opcionalmente, puedes guardar los diferentes informes en varias carpetas. Por ejemplo "reports-25u-10r-5i" es el nombre de la carpeta en la que guardamos los resultados de ejecutar el plan con 25 usuarios (hilos), un periodo de subida (ramp-up) de 10 segundos y 5 iteraciones.

Haz una última prueba: deshabilita todos los listeners excepto Simple Data Writer. Configura el grupo de hilos con 25 usuarios, un periodo de subida de 10 segundos y número de iteraciones infinito. Espera durante un minuto o así antes de parar la ejecución. Vuelve a generar los informes desde el terminal. Comprueba que hay 25 usuarios activos y luego consulta la gráfica *Charts→Over Time→Response Times Over Time* para ver si algún sampler tiene un tiempo de respuesta superior a 2 ms.

Al finalizar el ejercicio tu plan de pruebas debe tener el aspecto mostrado en la imagen de la derecha (en dicha imagen estan todos los listeners habilitados, tú debes verlos todos deshabilitados excepto el listener *Simple Data Writer*):



Plan de pruebas al finalizar el ejercicio

➪ ➪ ANEXO: Observaciones a tener en cuenta sobre la práctica P08

POM.XML

- Todos los tests se ejecutan con el plugin ***surefire*** (es el único plugin que necesitáis, además del plugin ***maven-compiler-plugin***)
- En el pom únicamente debéis tener las dependencias ***junit-jupiter-engine***, y ***selenium-java*** (está en las transparencias)

TESTS JUNIT

- Tenéis que usar en todas las clases que contienen los tests un método anotado con @BeforeEach y otro con @AfterEach (este último para cerrar el navegador)
- Si usáis un temporizador implícito, sólo debe aparecer una vez (en cada test, por lo que debe estar en el @BeforeEach).
- TODOS los assert se implementan en los tests, no en las PO

EJERCICIO 1:

- Los tests contienen código webdriver. Se implementan en src/test/java y no es necesario implementar nada en src/main/java
- El método *submit()* sólo lo usaremos para los formularios. Dicho método envía los datos al servidor y espera a que se cargue la nueva página. NO uséis el método click() para pulsar sobre el botón de un formulario (este método no espera a la carga de la nueva página).

PARA LOS EJERCICIOS 2 y 3:

- Las page object NO se instancian en el código del driver, excepto la primera page object. El resto de páginas se crean en el código de las page object, como resultado de la ejecución de sus métodos, tal y como se indica en las transparencias de teoría (ver tr. 23 y 24)
- Todas la PO se implementan en src/main/java
- Recuerda que los métodos de cada PO implementan un SERVICIO al que accedemos a través de una página html (en las transparencias de teoría tienes un ejemplo). No se trata de tener un método para cada WebElement!!! Por ejemplo, el servicio de login se implementa usando tres WebElements (dos cuadros de texto + el botón para enviar los datos). El objetivo es mejorar significativamente la mantenibiidad de los tests.

EJERCICIO 2:

- Los tests NO contienen código webdriver que dependa del código html.
- Los atributos de las clases que representan nuestras Page Object (PO) son únicamente de tipos de la librería Webdriver.
- Cada elemento de tipo WebElement debéis localizarlo preferentemente por id. Usa css antes que xpath. Los *locators* css y xpath puedes obtenerlo de forma automática a través del inspector del navegador, no obstante, intenta usar alguno de los patrones vistos en clase de teoría. Si usas el valor generado automáticamente, debes entenderlo igualmente.
- Las instancias de Page Object se crean con new(),
- En la clase que contiene los tests, NO debéis usar más atributos de los necesarios, .que básicamente son el objeto de tipo WebDriver, y los objetos que representan las Page Objects.

EJERCICIO 3:

- Los tests NO contienen código webdriver que dependa del código html, eso significa que tampoco pueden contener objetos de tipo Alert.
- Los atributos de las clases que representan nuestras Page Object son únicamente de tipos de la librería Webdriver.
- Las Page Object se crean siempre con la clase PageFactory, y sus atributos están anotados con @FindBy.

Resumen

¿Qué conceptos y cuestiones me deben quedar CLAROS después de hacer la práctica?

DISEÑO DE PRUEBAS DE ACEPTACIÓN DE PROPIEDADES EMERGENTES NO FUNCIONALES

- Los comportamientos a probar se seleccionan teniendo en cuenta la especificación (caja negra)..
- En general, las propiedades emergentes no funcionales contribuyen a determinar el rendimiento (performance) de nuestra aplicación, en cuyo caso tendremos que tener en cuenta el "perfil operacional" de la misma, que refleja la frecuencia con la que un usuario usa normalmente los servicios del sistema.
- Es muy importante que las propiedades emergentes puedan cuantificarse, por lo que deberemos usar las métricas adecuadas que nos permitan medir dichas propiedades.

AUTOMATIZACIÓN DE PRUEBAS DE ACEPTACIÓN

- Usaremos JMeter, para implementar nuestros drivers sin usar código java..
- Las "sentencias" de nuestros drivers NO son líneas de código escritas de forma secuencial, sino que usaremos una estructura jerárquica (árbol) en donde los nodos representan elementos de diferentes tipos (grupos de hilos, listeners, samplers, controllers...), que podremos configurar.. El orden de ejecución de dichos elementos dependerá de dónde estén situados en la jerarquía. Nuestro driver tendrá como nodo raíz un "plan de pruebas",
- Los "resultados" de la ejecución de nuestros test JMeter, consisten en una serie de datos calculados a partir de ciertas métricas (número de muestras tiempos de ejecución,...), que necesariamente estarán registradas en los listeners que hayamos usado para la implementación de cada driver.
- Los resultado obtenidos por la herramienta JMeter no son suficientes para determinar la validez o no de nuestras pruebas. Será necesario un análisis posterior, que dependerá de la propiedad emergente que queramos validar para poder cuantificarla.