UNIVERSITAT POLITÈCNICA DE VALÈNCIA

TRABAJO DE FIN DE GRADO

OPEN TEST SUITE

SUITE OPEN SOURCE DE AUTOMATIZACIÓN DE PRUEBAS DE REGRESIÓN PARA ENTORNO WEB

07 DE MARZO DE 2016

AUTOR DEL PROYECTO
ROBERT KOSZEWSKI

PROFESOR ASIGNADO
PAU MICÓ

0. Índice

1. Introducción

Cometer errores es de humanos. Esto también se aplica en el ámbito de la programación. Hasta el mejor programador del mundo puede cometer errores.

Además de esto, los programas suelen estar estructurados en módulos y funciones, que permiten reusar partes del programa y evitar de éste modo la duplicidad de código que realiza la misma tarea, facilitando enormemente la tarea de mantenibilidad y eficiencia.

1.1. Modularidad y actualizaciones

Gracias a ésta modularidad, se permite a más de un programador trabajar sobre el mismo proyecto al mismo tiempo, como también hacer uso de módulos de terceros prefabricados que resuelven un cierto problema sin necesidad de que el programador tenga que implementar esa parte de nuevo con el respectivo impacto en el tiempo de desarrollo para el programa final.

Éstos módulos se suelen ir actualizando con el tiempo según las necesidades del proyecto y para solucionar errores también conocidos como *bugs*. Éstas actualizaciones a veces terminan cambiando ligeramente el comportamiento del módulo en comparación al que tenía al principio, realizando la misma tarea, pero teniendo un resultado diferente al que hubiera tenido anteriormente con las mismas condiciones. En ciertos casos incluso se opta por eliminar ciertas funciones o módulos porque se entiende que está obsoleta o no se usa en la aplicación.

1.2. Los errores en los programas

Los errores en los programas siempre pueden pasar y es responsabilidad del desarrollador de proporcionar un cierto grado de calidad para el programa asegurando su correcto funcionamiento. Esto a medida que la solución se vuelve más compleja resulta ser una tarea cada vez más difícil y requiere de una organización más compleja.

Publicar un programa con errores no es deseado ya que se considera defectuoso y puede frustrar al usuario final, generar desconfianza y proyectar una mala imagen sobre la empresa o desarrollador que lo ha publicado, llegando incluso a afectar lo económicamente.

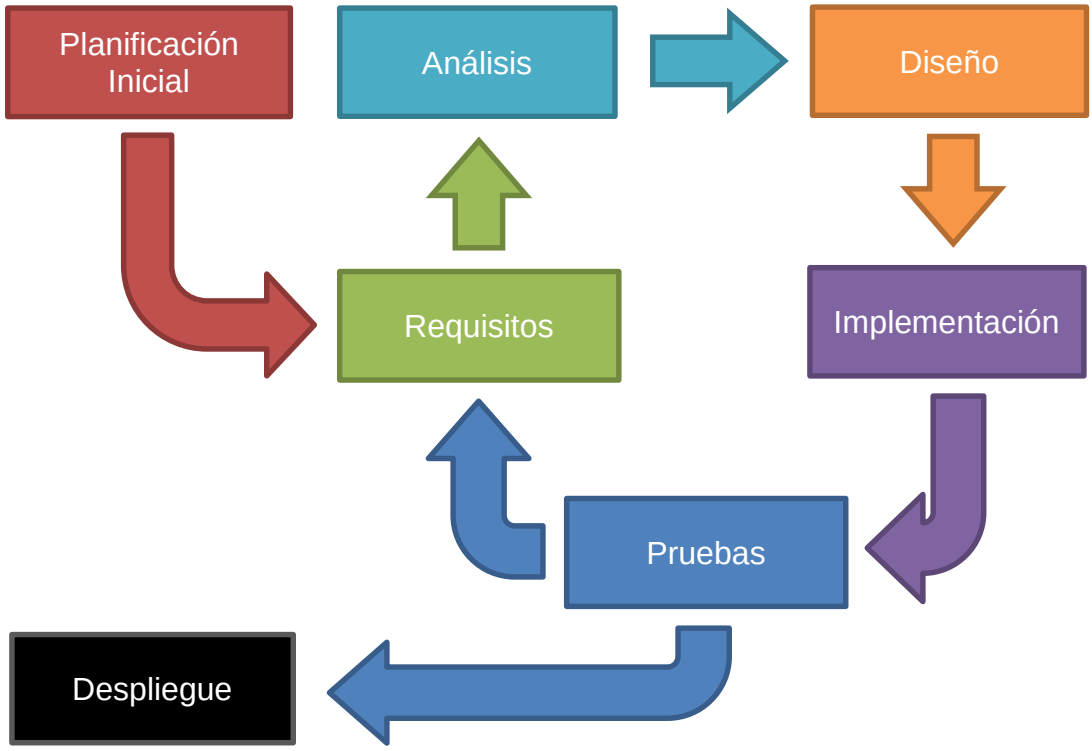
De ahí que antes de poder dar un programa a la disposición del usuario final, se ha de realizar una serie de pruebas para asegurarse de que la aplicación funciona correctamente y que presenta una cierta calidad. De lo contrario se deberán solucionar los errores si los hay algunos presentes hasta mitigarlos completamente.

Cuando un programa es de poca extensión y es relativamente sencillo, el programador suele ejecutar el programa y comprobar que todo funciona como se espera antes de publicar el programa. Sin embargo, a medida que el programa crece en extensión y complejidad, esta tarea se vuelve más difícil y requiere de una planificación más adecuada.

1.3. Pruebas de software

Es el último paso en el ciclo del desarrollo antes de ser publicado. Las pruebas de software tienen por objetivo probar que las soluciones desarrolladas cumplan con las funciones específicas para los cuales han sido creados. Existe una gran variedad de tipos de pruebas que van desde pruebas estáticas que se realizan sobre el código, pero sin ejecutarlo, y pruebas dinámicas, que realizan las pruebas sobre el programa ejecutándose.

A continuación, se puede ver un esquema del desarrollo iterativo en el que se puede ver que el paso de “Pruebas” es en el que se decide si la aplicación está lista para ser desplegada o pasa a la siguiente iteración de la producción.



Éste documento se centrará en las pruebas dinámicas, que son las más comunes.

1.4. Pruebas de caja blanca

Las pruebas de caja blanca se centran en probar cada función y componente de la aplicación, y comprobar que el resultado devuelto por el mismo es el esperado. Éste enfoque está fuertemente ligado al código fuente y requiere que la persona encargada de realizar las pruebas (*el tester*) esté familiarizada con la misma. Éste modelo sin embargo se limita a comprobar las diferentes partes del sistema, pero no permite comprobar el conjunto con su usabilidad, como un usuario final lo haría.

1.5. Pruebas de caja negra o funcionales

Las pruebas de caja negra o funcionales se centran en realizar las pruebas desde el punto de vista del usuario final, tomando al programa como una caja negra en la que no se sabe cómo trabaja por dentro (no se requiere

conocimiento del código fuente), limitándose simplemente a interactuar con la aplicación mediante sus métodos de entrada, que suele ser la interfaz de usuario, y retroalimentándose con los resultados que éste devuelve.

Existen numerosas pruebas de caja negra o funcionales, pero éste proyecto se centrará principalmente en las **pruebas de regresión**.

1.6. Pruebas de regresión

El objetivo de las pruebas de regresión es la de verificar que el software que ya ha sido desarrollado y testeada previamente, sigue trabajando de la forma esperada después de ser actualizado o modificado. Como efecto onda, a veces la modificación de una parte del software, afecta a otra parte del programa que no ha sido modificada, pero su funcionamiento se puede ver afectado o corrompido. Éste tipo de pruebas intenta descubrir lo que ha dejado de funcionar que antes sí funcionaba. La forma más común de realizar las pruebas de regresión es realizando una serie de pasos y comprobando que el resultado es el mismo como anteriormente.

1.7. El problema de las pruebas de aplicaciones cada vez más grandes

A medida que la aplicación se hace más grande y crece en funcionalidades, la tarea de probar la aplicación manualmente se hace más larga y tediosa con cada iteración, llegando a un punto que ya no resulta viable. Sin embargo, éste proceso se puede automatizar usando software de automatización.

2. Software de automatización de pruebas de regresión

Al usar software te automatización se puede prevenir tener que realizar tareas repetitivas una y otra vez de forma manual, cuando se pueden realizar por software totalmente automatizado y más rápidamente.

El tester en vez de tener que realizar las pruebas manualmente, se dedica programar una serie de pasos que simulan la interacción humana con el software, y éstas son reproducidas automáticamente cuando son requeridas. De éste modo se elimina la necesidad de que el tester tenga que volver a realizar los mismos pasos, y sólo se dedique a actualizar los pasos para las nuevas funcionalidades del programa.

Dada la gran variedad de programas, en éste proyecto me centraré específicamente en la automatización de pruebas de regresión para **web** y **web-apps**.

2.1. Soluciones comerciales existentes

Existe una variedad de soluciones comerciales existentes que facilitan la automatización de pruebas funcionales para web, sin embargo, éstas suelen tener un precio elevado, que rondan los precios a partir de 2.500€ al año o al mes, siendo todas por suscripción sin opción de pago único. Todo esto sirve como barrera para empresas más pequeñas o que no desean realizar una inversión tan importante en éste tipo de software.

Una breve lista de soluciones comerciales:

- HP Unified Functional Testing: **http://goo.gl/aJgt7i**
- Telerik Test Studio: **http://goo.gl/yMdV7s**
- IBM Rational Functional Tester: **http://goo.gl/rgMhK7**
- SahiPro: **http://goo.gl/kXC3XW**

2.2. Soluciones gratuitas u open source existentes

Todas las soluciones gratuitas u open source existentes suelen ser incompletas, desfasadas o las que son completas van dirigidas a usuarios con conocimientos en programación usando lenguajes como Java, Python y JavaScript para programar las tareas.

Una breve lista de soluciones gratuitas u open source:

- Selenium: **http://goo.gl/TROC8v**
- AutoIt: **https://goo.gl/l2I6y0**
- Sahi (Open Source): **http://goo.gl/T9fP8C**

3. Motivaciones

Dado que no hay ninguna solución fácil y decente para realizar pruebas funcionales para entornos web, sería interesante disponer de una solución gratuita y a ser posible open source que permita escribir pruebas funcionales y que no requiera de habilidades de programación.

3.1. Usuarios no expertos en programación

Éste último punto es importante, ya que la gran mayoría de usuarios finales de los programas que se desarrollan no suelen ser técnicos informáticos ni programadores, en los departamentos de pruebas (QA) no se suelen emplear personas con gran conocimiento en programación para tener un mejor punto de vista del usuario final. De ahí que una solución que sea lo suficientemente simple e intuitiva que no requiera conocimientos de programación sería de gran utilidad.

4. Solución al problema

La idea principal es hacer que la solución se pueda usar en una gran variedad de entornos, y dado al auge reciente de plataformas de tipo **ARM**, sería interesante disponer de un sistema que dé soporte a ésta, como también a la plataforma **x86** que es la más usada actualmente. Éstos requisitos se cumplen con la tecnología **Java** con lo que la implementación será principalmente en **Java** y con soporte para los tres sistemas operativos más usados actualmente (**Windows**, **Linux** y **Mac**) en plataformas **ARM** y **x86**.

4.1. Las 3 tareas fundamentales

El programa implementará soluciones para las 3 principales tareas que son **crear/editar tests**, **ejecutar los tests** y **generar estadísticas** a partir de los resultados obtenidos durante la ejecución del test. Una lista detallada de las 3 tareas fundamentales:

1. **Ejecutar los Tests:** Una interfaz estándar (Sea GUI, Consola, Web, etc) que permite ejecutar y mostrar el estado del test ejecutándose de forma clara e intuitiva.
2. **Crear y/o Modificar Tests:** Una interfaz estándar organizado e intuitivo que permita crear y modificar las diferentes tareas soportadas por los módulos.
3. **Generar Informes:** Una interfaz que permita mostrar los datos recopilados durante los Tests, generando informes según sea necesario.

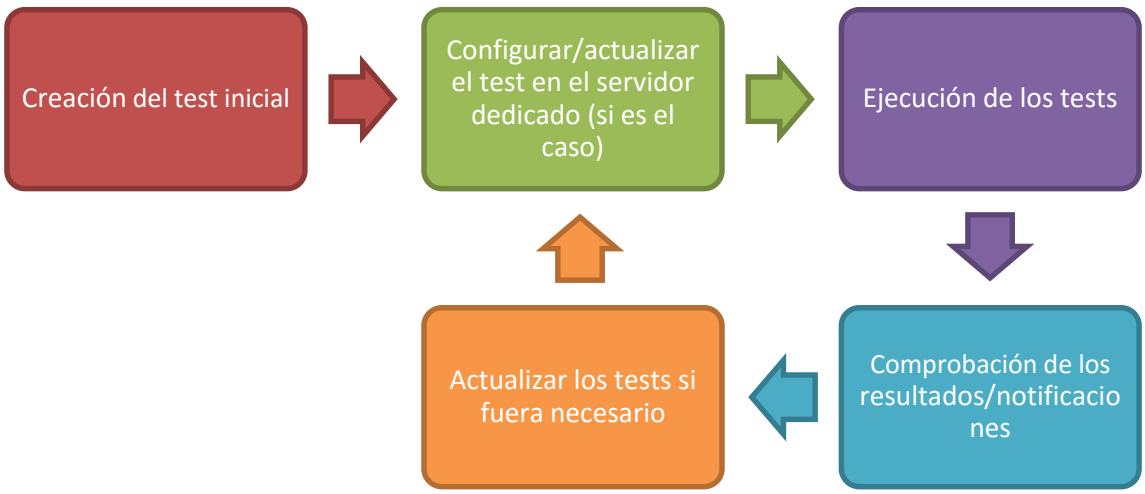
4.2. Los 3 modos principales de ejecución

Principalmente se usará una interfaz de usuario (GUI) local para **crear**, **editar**, **ejecutar** y **obtener informes** de los resultados. Sin embargo, también es interesante tener una posibilidad de controlar el programa desde la consola, para entornos que no dispongan de interfaz de usuario, o para posibilitar el control remoto mediante **Telnet** o **SSH**. Finalmente, una interfaz web que pudiera ser usado por más de una persona a la vez sería de gran utilidad para servidores dedicados al *testing* que puedan ser configuradas y programadas por un equipo dedicado al testing. En un breve vistazo se dispondrán de los siguientes 3 modos:

1. **Modo Normal (Local):** El modo principal que permite a un usuario en su entorno local realizar las 3 principales tareas.
2. **Modo Consola (Local y/o Remoto):** Un modo simplificado sin interfaz de usuario, que permita controlar el programa desde la consola.
3. **Modo Web (Remoto):** Posibilidad de configurar y ejecutar tests en un servidor dedicado a ello.

4.3. Flujo de trabajo

Se intenta mantener un flujo de trabajo de creación de tests lo más sencillo posible. Se parte de la creación de un test inicial. En el caso de existir servidor dedicado de testing configurarlo, ejecutar los tests, comprobar los resultados o ser notificado de ello y si fuera necesario actualizar los tests.



4.4. El concepto de acciones o "Actions"

Dado que la aplicación va dirigida a usuarios no expertos en programación, se ha diseñado el concepto de acciones o "**Actions**" que representan las acciones que se desean realizar en relación a un eje temporal. De éste modo el usuario no tiene que tener conocimientos de programación y su trabajo se basa en ordenar acciones de tipo "*HACER CLICK EN..*", "*ESPERAR A..*", "*COMPROBAR QUE..*", etc. Éstas acciones son ejecutadas una detrás de otra manteniendo un orden temporal. Las acciones son sin embargo lo suficientemente poderosas que permite a usuarios más avanzados usar condiciones, saltos condicionales, variables y configuraciones en tiempo real durante la ejecución de los tests permitiendo un control prácticamente total sobre lo que sucede en cada momento.

4.5. Herramientas y entorno de desarrollo

Para el desarrollo se ha optado por usar Eclipse (**https://www.eclipse.org**) con los siguientes módulos y extensiones:

- **Java Development Tools (JDT):** Conjunto de herramientas para el desarrollo en Java.
- **Web Development Tools (WDT):** Conjunto de herramientas para el desarrollo de páginas web y edición de JavaScript.
- **eGit:** Módulo que integra la plataforma GIT en Eclipse
- **JasperReports Helper Tools:** Herramienta para la creación y edición de informes en formato *jasper*.

4.6. Licencia y publicación

Open Test Suite se publica bajo licencia open source **GNU GPL v2** que permite el uso, modificación y extensión de la aplicación, siempre que no se reste crédito al autor. La extensión y modificación de la aplicación deberá ir bajo la misma licencia y deberá ser acompañada siempre con el código fuente incluso si es parte de un paquete comercial.

El diseño, logos e iconos de Open Test Suite (Sin contar los iconos usados para las acciones y botones) han sido diseñados específicamente para la aplicación con lo que son amparados bajo la misma licencia.

La descarga del programa final y más información sobre el mismo se puede encontrar de la página web del proyecto:
http://opentestsuite.robertkoszewski.com

El código fuente puede ser encontrado en BitBucket en la siguiente dirección:
https://bitbucket.org/RKKoszewski/opentestsuite

La tecnología usada para la gestión del código fuente es **GIT** y desde la misma web se puede obtener todo el código fuente como también realizar *forks*.

5. Implementación

Se ha empezado primero implementando un Prototipo que se ha puesto a disposición a un equipo de testing (*QA*) en una empresa real para comprobar la viabilidad de éste proyecto y para obtener *feedback* sobre posibles mejoras y así concluir el producto final.

5.1. El Prototipo

El programa se puede ejecutar en 3 modos diferentes según los datos que se le especifican al lanzarlo (**Modo interfaz**, **modo consola** y **modo web**) que se describirán más adelante.

La aplicación principal está dividida en 5 partes que son separados mediante pestañas (Ver apartado de Navegación). Todos los módulos son inicializados y cargados durante el inicio de la aplicación.

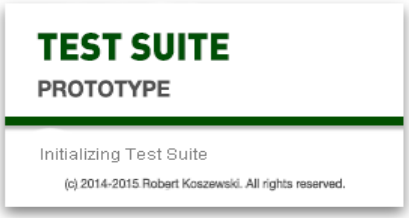
5.1.1. Navegación

El prototipo actual del programa separa los diferentes módulos (o partes) del programa en pestañas como se puede ver en la siguiente imagen.

Para asegurar que usuarios nuevos no se pierdan en la funcionalidad en la aplicación, se ha implementado bocadillos de información sobre prácticamente la totalidad de componentes que se pueden encontrar en la interfaz, explicando que hace cada componente.

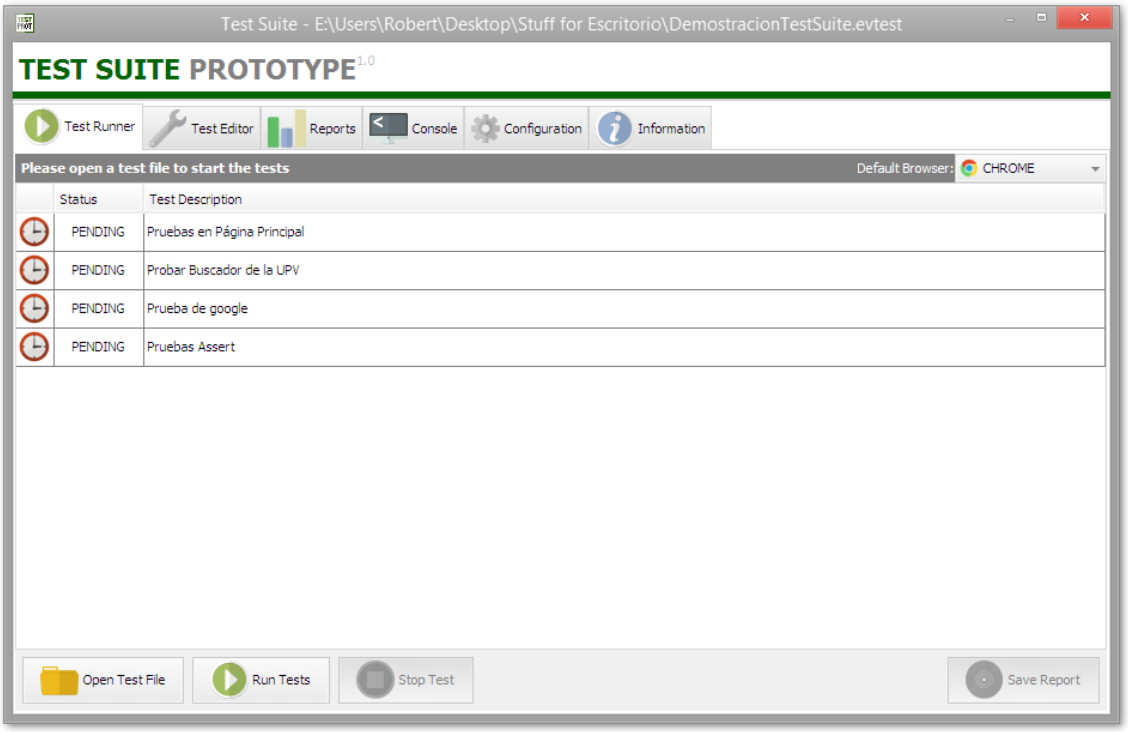
5.1.2. Inicio de la aplicación

Cuando se ejecuta la aplicación se muestra una imagen *splash* que muestra que la aplicación se está iniciando y su progreso (Esto es especialmente útil en entornos dónde el programa tarda mucho en iniciarse o el antivirus de la empresa lo ralentiza tanto que el usuario piensa que el programa no se ha iniciado).



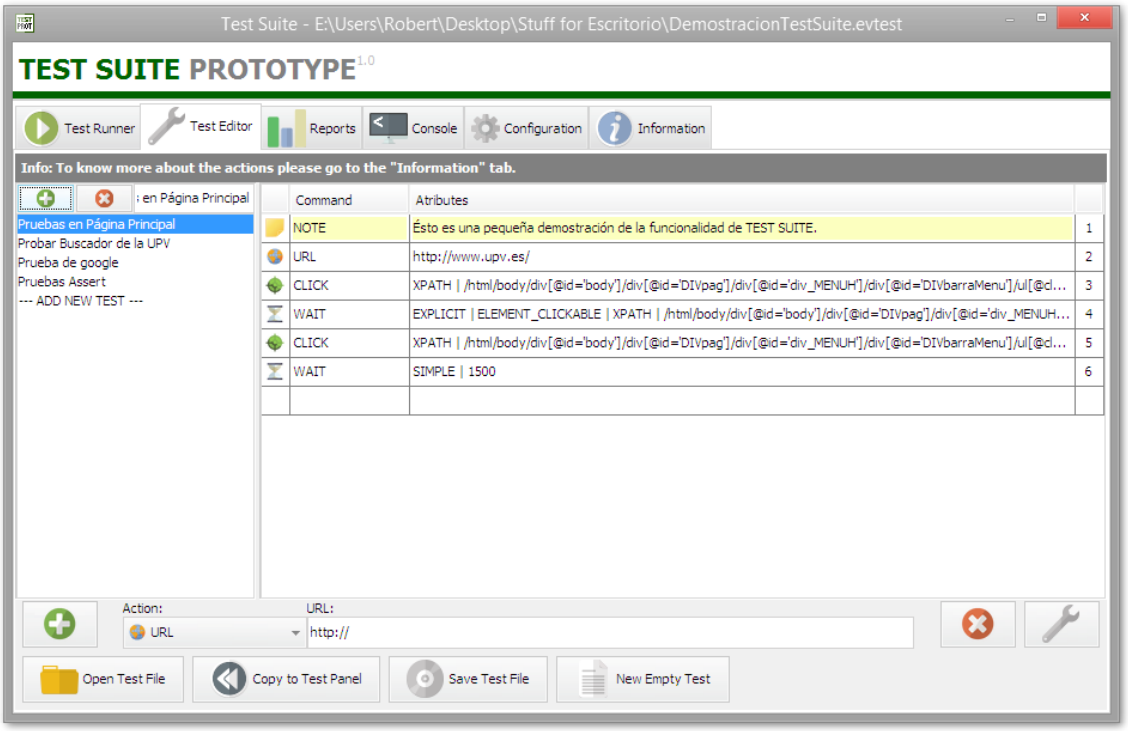
5.1.3. Interfaz Principal – *Test Runner*

Ésta es la primera interfaz que se abre tras cargar la aplicación. Es la llamada "**Test Runner**" que permite ejecutar los diferentes tests. Presionando sobre "**Open Test File**" se permite seleccionar y cargar un test guardado. Presionando sobre "**Run Tests**" se ejecuta el test cargado. Durante la ejecución del test, se puede ejecutar "**Stop Test**" para parar el test. Una vez finalizado un test se puede presionar sobre el botón "**Save Report**" para guardar el resultado del test en un archivo de formato **JSON**. Se puede cambiar el navegador por defecto cambiando lo en el desplegable "**Default Browser**", por defecto es seleccionado el navegador especificado en el archivo de test.

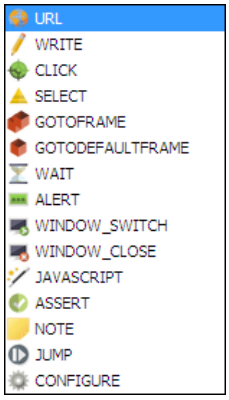


5.1.4. Test Editor

En éste apartado se puede crear y editar los diferentes tests. Los tests están agrupados en grupos de comandos (Llamadas *Actions*) que son ejecutados en orden descendiente uno detrás de otro con excepción del comando **JUMP** que permite saltar a otro comando según una determinada condición, permitiendo implementar lógica de tipo **IF-ELSE**, **WHILE**, etc.



Se pueden **añadir**, **modificar**, **quitar** **comandos**. **Crear**, **modificar** y **quitar** **grupos completos**. Se soporta el **copiado** y **pegado** de una o varios comandos y/o grupos enteros, incluso entre instancias diferentes del programa. Ambos se pueden mover de posición mediante **ALT+Flechas de dirección**. "**Open Test File**" permite abrir un archivo de test existente. "**Copy to Test Panel**" copia el test actual a "**Test Runner**" sin necesidad de guardarlo previamente. "**Save Test File**" permite guardar el test a un archivo. "**New Empty Test**" elimina por completo el test actual e inicia uno nuevo vacío. Apretando sobre la herramienta se puede acceder al *menú avanzado del test*, que se describirá posteriormente. El editor completo se puede manejar con tan solo el teclado usando el botón de tabulador.

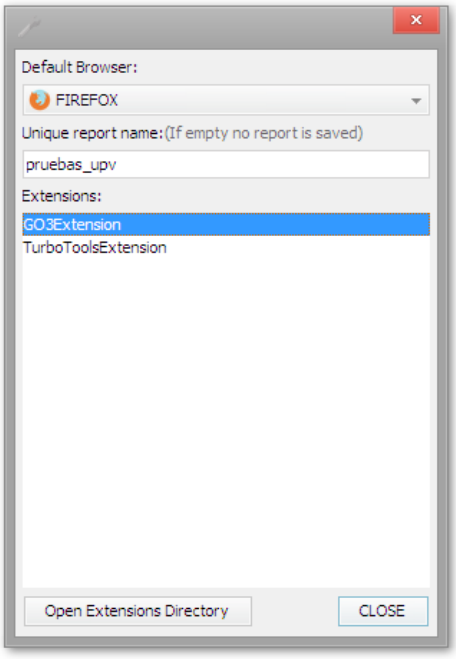


Los comandos disponibles por defecto son los que se pueden ver en la imagen de la izquierda. El número de comandos se puede ampliar mediante extensiones que se pueden habilitar en el *menú avanzado del test*.

Cada comando puede tener opciones diferentes que se muestran de forma dinámica según son seleccionadas.

Una lista parcial de comandos y sus opciones se mostrará posteriormente en éste documento.

5.1.5. Test Editor – Menú avanzado del test



Desde aquí se pueden gestionar, habilitar/deshabilitar las diferentes extensiones, seleccionar el navegador que se usará por defecto en el test y un nombre para la base de datos que almacenará los resultados del test. Éste deberá ser único si no se desea que se mezclen los resultados. Esto es especialmente útil si en un futuro se amplían los *tests*, permitiendo seguir disponiendo de los datos de versiones anteriores de *tests* en los informes.

Las extensiones permiten añadir nuevas **Actions** que pueden ser usados desde el editor de tests. Las extensiones se habilitan por test, es decir, que cuando se crea un nuevo test en blanco, por defecto todas las extensiones quedan deshabilitadas. Para poder ejecutar los tests que usan extensiones personalizadas en otra máquina, se tendrá que instalar las extensiones requeridas en la nueva máquina. Sin embargo, el programa es capaz de cargar tests con extensiones no instaladas, en los casos donde se han habilitado una o más extensiones, pero nunca se han usado en toda la test suite.

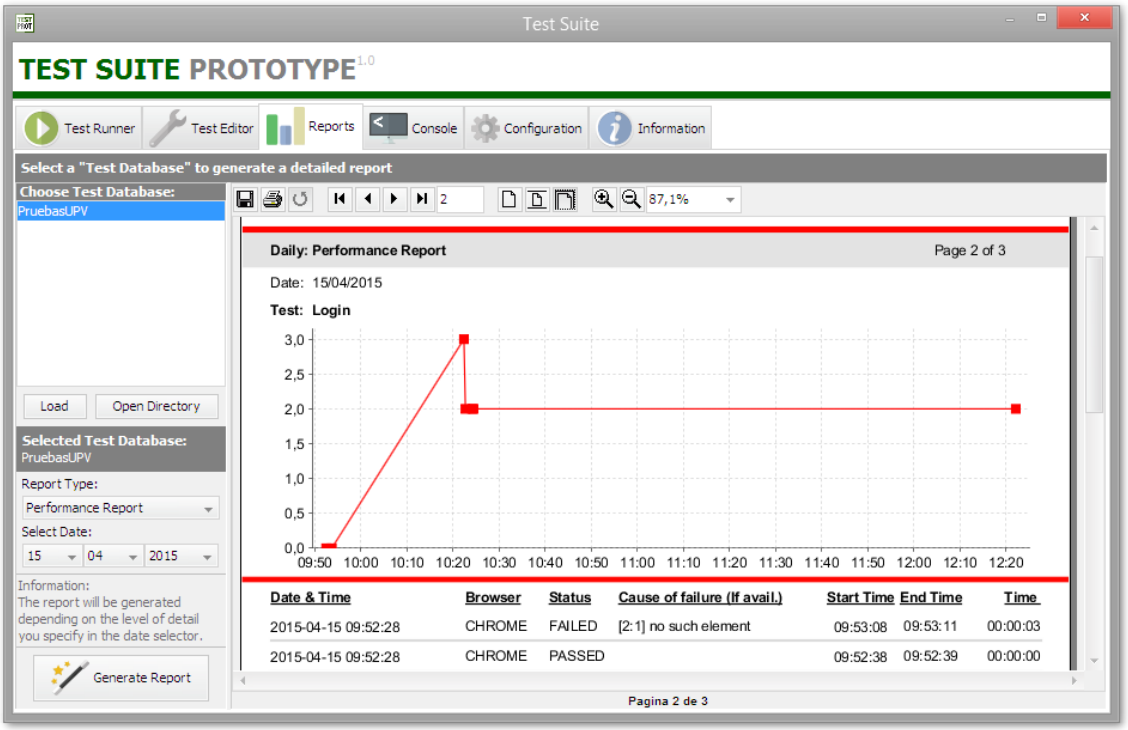
5.1.6. Reports

Desde aquí se pueden generar informes a partir de las bases de datos de los *tests*. Estos se pueden **eliminar/añadir/copiar** abriendo el directorio presionado sobre "**Open Directory**".

Una vez cargada la base de datos que se desea, se pueden generar informe tipo "**Performance**" (Velocidad de carga) y de tipo "**Test-Fail**" de resultados de fallos de las pruebas. Todo con posibilidad de filtrado por **Año/Mes/Día**.

Según se seleccione se puede generar un informe de mayor detalle (Diario) o más general (Vista mensual o anual).

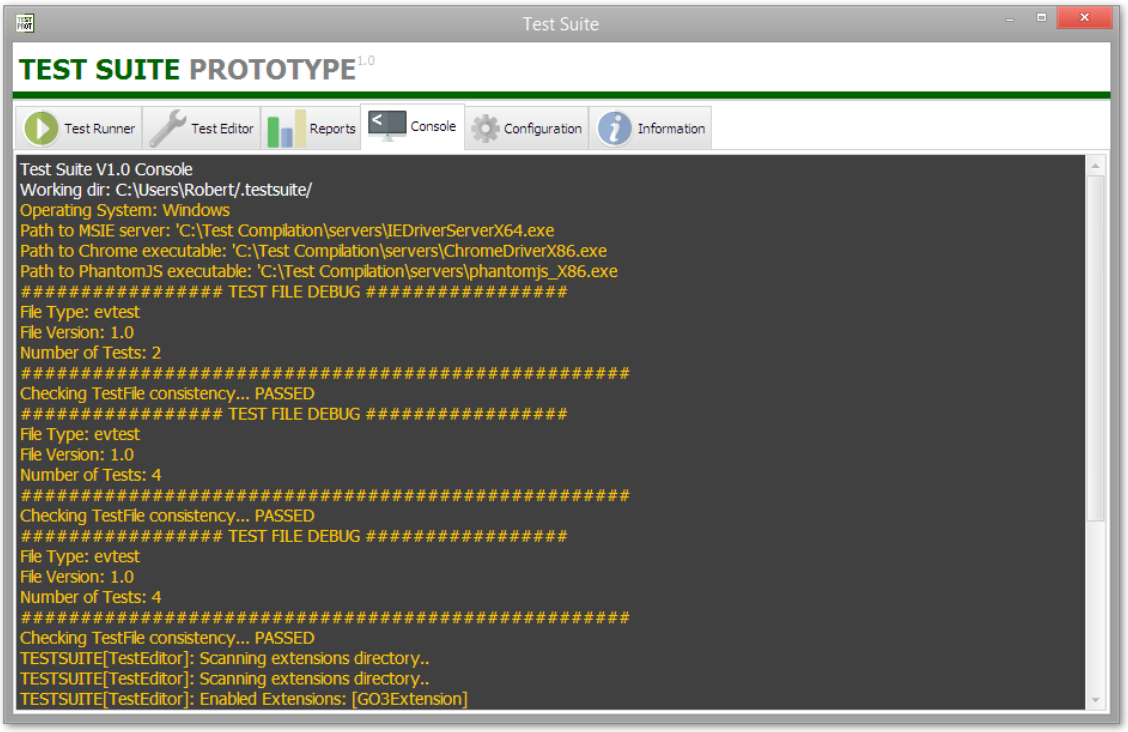
A continuación, se puede ver la presentación en la interfaz del programa:



Los informes generados se pueden visualizar directamente en el programa, pueden ser impresos o guardados en una gran variedad de formatos (Incluyendo PDF, ODT, DOCX, HTML, etc).

5.1.7. Console

En ésta sección se puede consultar la consola que muestra información de tipo *debug* y de estado.



Este apartado puede resultar confuso para la mayoría de usuarios no técnicos, aunque resulta muy útil para usuarios más avanzados para obtener más información cuando algo está fallando.

5.1.8. Configuración

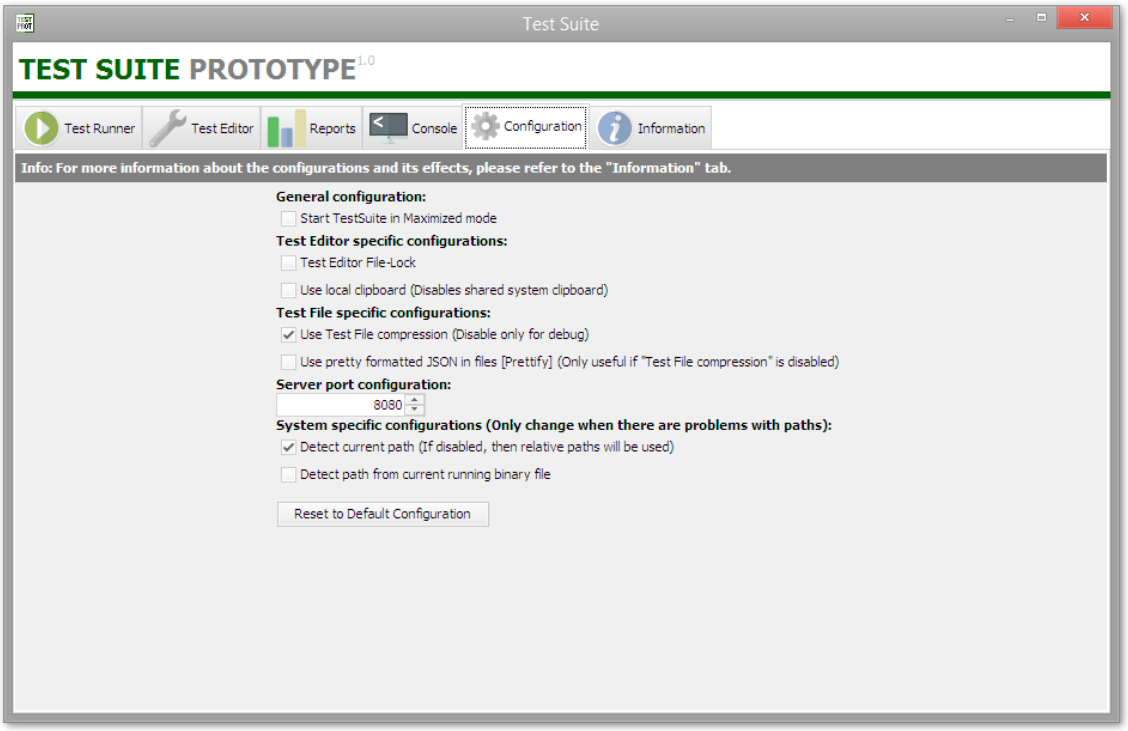
En éste apartado se pueden configurar ciertas funcionalidades de Test Suite. Entre las opciones disponibles está la posibilidad de abrir Test Suite en **modo maximizado por defecto**.

Usar **File-Lock** mientras se edita un archivo (Esto es muy útil cuando se usan almacenamientos compartidos en red y hay varias personas encargadas de crear tests, de éste modo se previene que dos personas diferentes estén editando el mismo archivo de test al mismo tiempo, sin embargo, también previene otras instancias del programa editar un archivo ya abierto y puede confundir al usuario de ahí que viene desactivado por defecto).

Usar el **clipboard** local que limita el ámbito de la función de copiar y pegar del editor a únicamente la instancia local (No se puede copiar y pegar de una instancia de la aplicación a otra), esto trae como ventaja que cada aplicación tiene su propio **clipboard**, sin embargo, ésta opción viene deshabilitada por defecto.

Opciones para personalizar el formato del archivo de test, aunque no se recomienda cambiar la configuración por defecto ya que solo sirve para el propósito de **debuging**. Se puede cambiar el puerto del servidor cuando se inicia Test Suite en modo servidor.

Finalmente, dos configuraciones de compatibilidad en caso de haber problemas con el *Path* en los sistemas y una opción para restaurar la configuración por defecto.



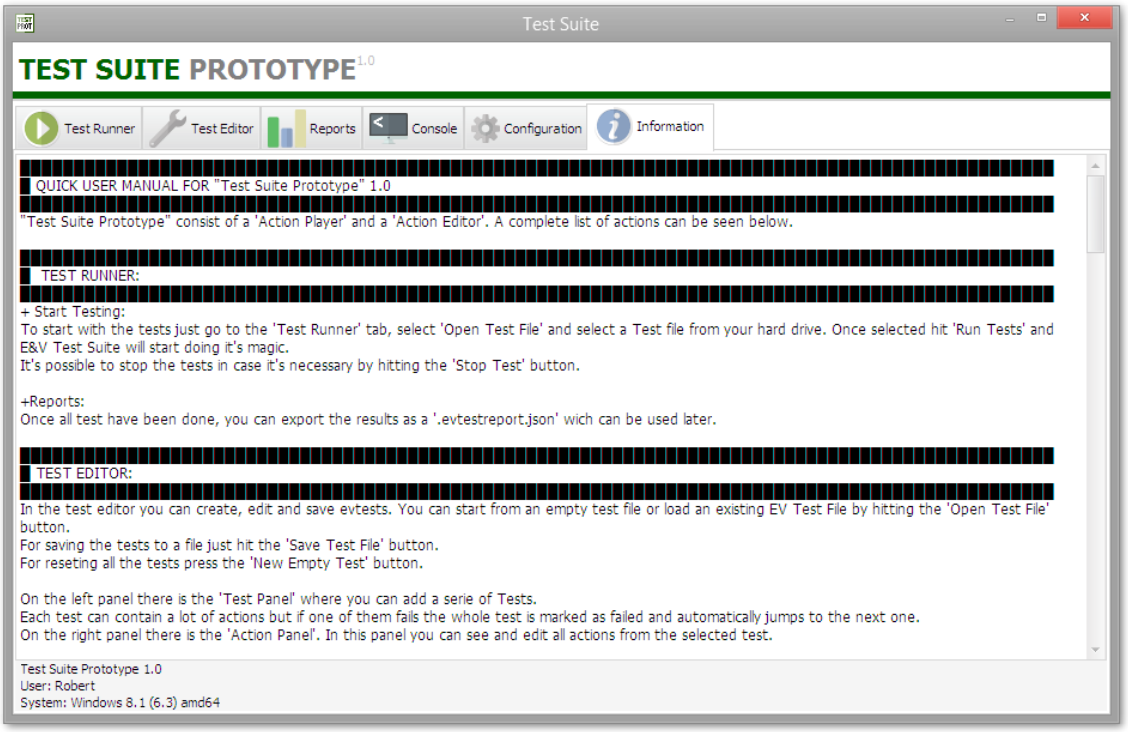
Todas éstas configuraciones se pueden ajustar dinámicamente cuando se lanza la aplicación.

Todas éstas configuraciones son explicadas con más detalle en el manual de usuario de **Open Test Suite** disponible en el anexo de éste documento.

Todas las opciones son explicadas con detalle mediante bocadillo, cuando se pone el ratón por encima de una opción en el programa.

5.1.9. Information

Éste apartado sirve básicamente como manual de usuario integrado en la misma aplicación. Trae información sobre prácticamente todas las funcionalidades de la aplicación incluidas sus descripciones.

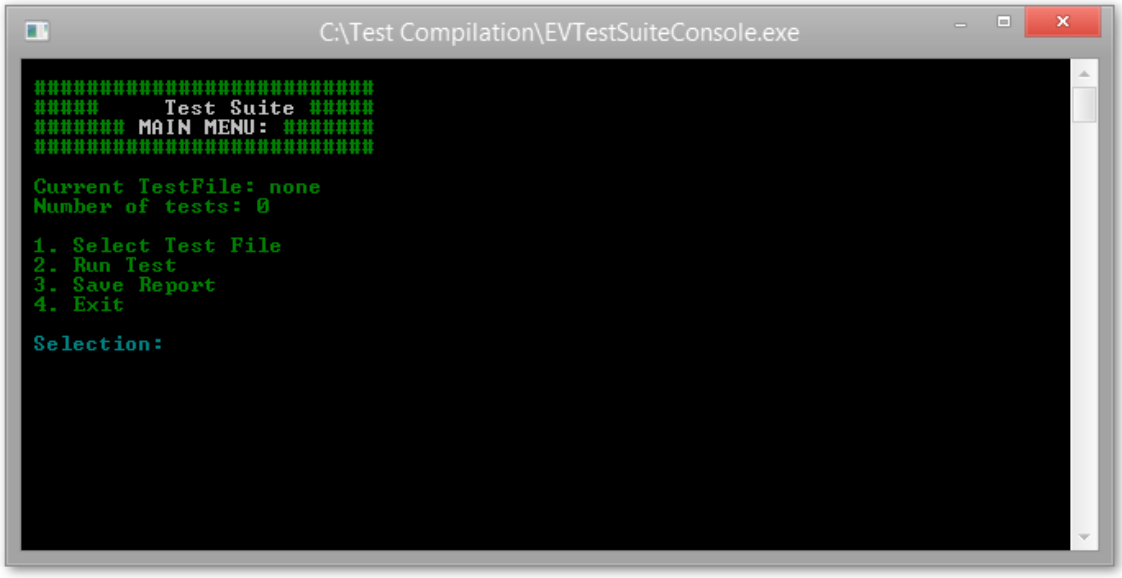


Además de servir como manual, se puede obtener información sobre la versión de Test Suite, el usuario actual y el sistema operativo actual para el caso en el que sea necesario.

El extracto del texto de éste apartado se mostrará al final de éste documento (Apartado Extracto – Manual de Usuario).

5.1.10. Modo – Console

El programa se ejecuta en modo **console** si al lanzarlo se especifica la opción "**-consolemode**". Desde aquí se pueden ejecutar los tests en modo consola, permitiendo realizar ésta tarea incluso de forma remota mediante telnet o SSH.

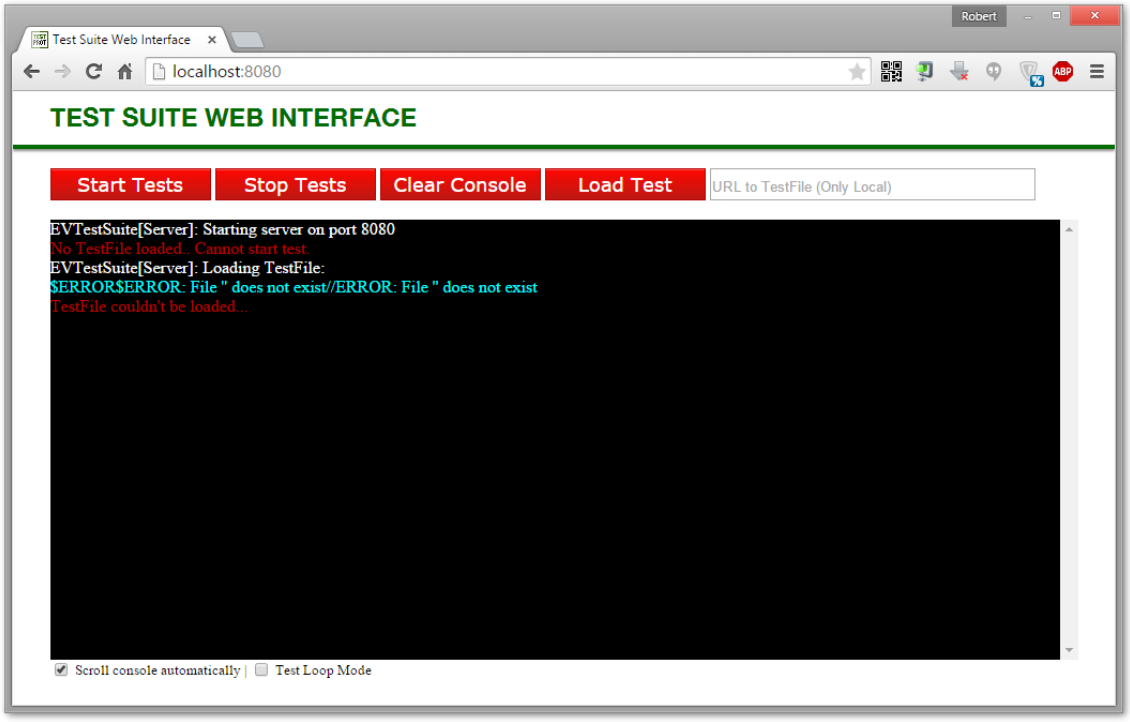


Especificando un archivo de test durante el inicio, ejecutará el test directamente e incluso mediante la combinación de opciones se puede lanzar una prueba sin que se muestre nada y cuando el test termina devolver un *JSON* con el estado del test que se muestra por pantalla. Esto es muy útil si se quiere hacer un servidor que ejecuta *tests* de forma automatizada y que recoja los resultados que pueden ser *parseados* por el mismo una vez terminado el test.

En éste modo también se evita cargar todo el resto de módulos que no sean el del "**Test Player**" con su consecuente impacto en la memoria RAM.

5.1.11. Modo – Web Server

El programa es iniciado en modo servidor si se especifica la opción "-server" durante el lanzamiento de la aplicación. En éste modo se permite al usuario conectarse al puerto especificado usando un navegador web de forma remota y poder así monitorizar, ejecutar y parar *tests* en la máquina del servidor.



Con la opción "**Test Loop Mode**" se ejecuta el test seleccionado de nuevo tras un breve tiempo de espera. Con esto se puede habilitar un servidor dedicado a las pruebas.

5.2. Puesta a disposición a un equipo real QA con su feedback

El prototipo se ha puesto a disposición a un equipo de tests (QA) de una empresa real, para poder observar la reacción de los usuarios con el programa, y para poder recibir feedback sobre su uso.

La acogida del programa ha sido bastante exitosa. El manual en la aplicación y el hecho de que prácticamente todos los botones y elementos que figuran en la aplicación vienen con explicaciones cuando uno pone el ratón por encima del elemento sobre el que quiere saber mayor información

5.2.1. Principales dificultades

- El desarrollo de tests de mayor longitud resulta tedioso incluso con los simplificados **Actions**, y los *testers* tienden a usar otras herramientas más limitadas pero que permiten la grabación directa desde el navegador, resultando en tests largos que funcionan, pero inútiles, ya que no pueden generar los resultados y estadísticas requeridos y no soportan entornos dedicados.
- La falta de unas normas de creación suele causar que los *testers* creen tests de poca calidad que fallan con mucha facilidad cuando cualquier pequeña circunstancia cambia.

5.2.2. Proposición de mejoras

- El equipo de testing (QA) ha propuesto que se considere la ampliación del modo servidor web, ya que sería muy interesante tener un servidor dedicado que ejecutara los tests periódicamente de forma automática y recibir algún tipo de notificación si algún tipo de problema estuviera sucediendo.
- Disponer de una interfaz oscura sería bastante útil para cuando se pasa muchas horas usando el programa, para no cansar tanto la vista de los colores claros.

5.3. Implementación final del programa

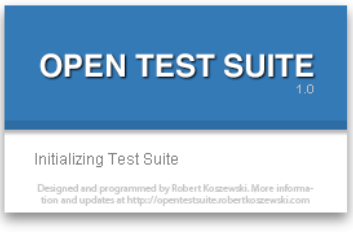
Tras el feedback del equipo de testing (QA) se ha procedido a implementar la versión final del programa. Dado el origen de código fuente del programa se procede a renombrar el programa a un nombre más acorde, **Open Test Suite**.

Las acciones se han diseñado para ser más perdonables cuando es posible, teniendo en cuenta los errores más comunes e intentando evitarlos.

Por otro lado, se ha añadido un sistema de compatibilidad con soluciones de terceros. Open Test Suite es capaz de importar tests grabados con "**Selenium IDE plugin for Firefox**" y convertirlos al formato nativo de la suite.

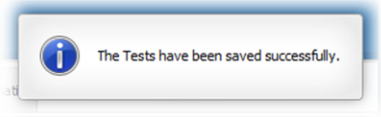
5.3.1. Inicio de la aplicación

El inicio de la aplicación se ha cambiado acorde al nuevo nombre y simplificado ligeramente, ocupando menos espacio y mostrando solo la información más relevante.



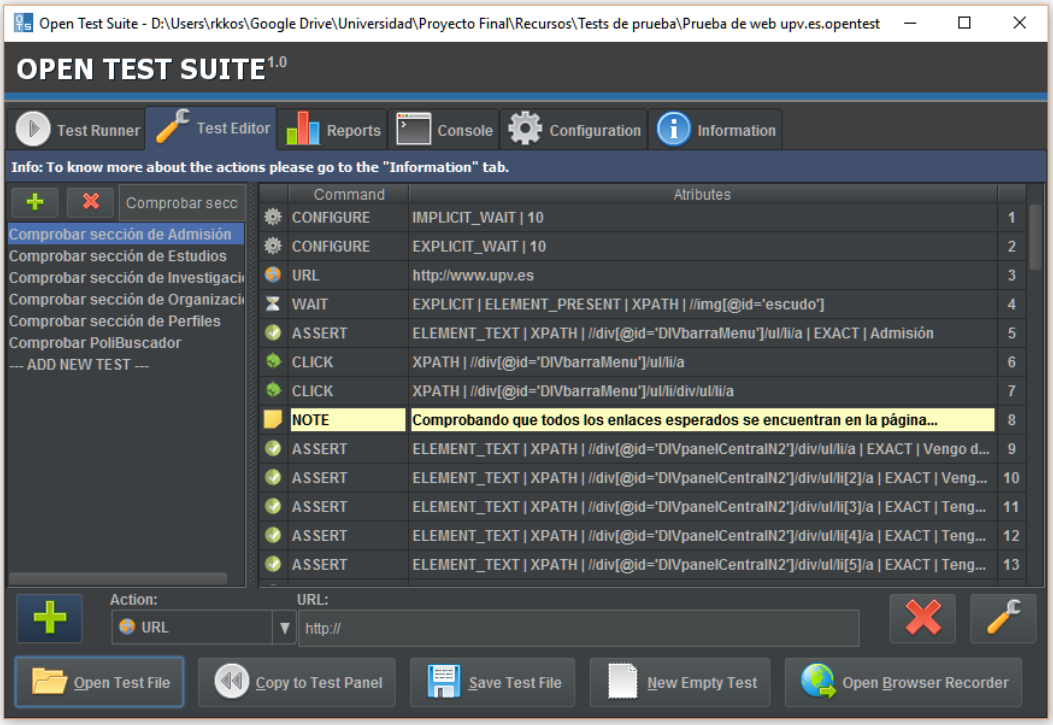
5.3.2. Diseño gráfico unificado y mejoras de usabilidad

Se han tenido en cuenta muchísimo en ésta versión las "*best practices*" en diseño en interfaces y usabilidad. El diseño en toda la aplicación es consistente en todo momento. Se han eliminado prácticamente por completo todos los mensajes con preguntas tipo "Seguro que quiere hacer eso?" y cambiándolos por la opción de deshacer lo que se ha hecho, mostrando un mensaje no intrusivo de que se ha realizado dicha acción.



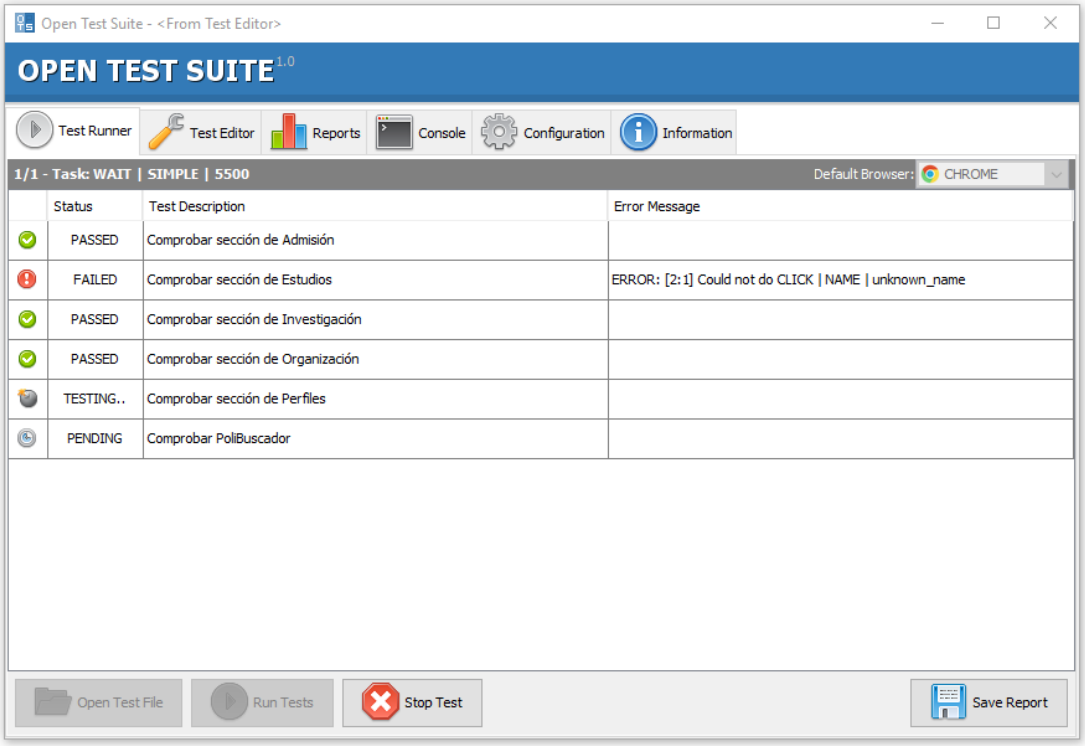
Se ha mejorado la usabilidad con el teclado, llegando a poder usar la aplicación completamente sin ratón. Por el contrario, también se ha mejorado el uso con ratón, permitiendo realizar acciones tipo *Drag&Drop*. La mayoría de iconos han sido cambiados a una variante menos borrosa y de licencia gratuita. También se ha añadido una animación de carga en la aplicación cuando se necesita más tiempo del normal para realizar una tarea sin respuesta en la interfaz. Anteriormente la interfaz se quedaba congelada y parecía que el programa estaba bloqueado.

También se ha añadido una versión oscura del diseño de la aplicación, para evitar el cansancio ocular durante temporadas prolongadas de uso del programa. A continuación se puede ver una captura de pantalla del diseño oscuro del programa:



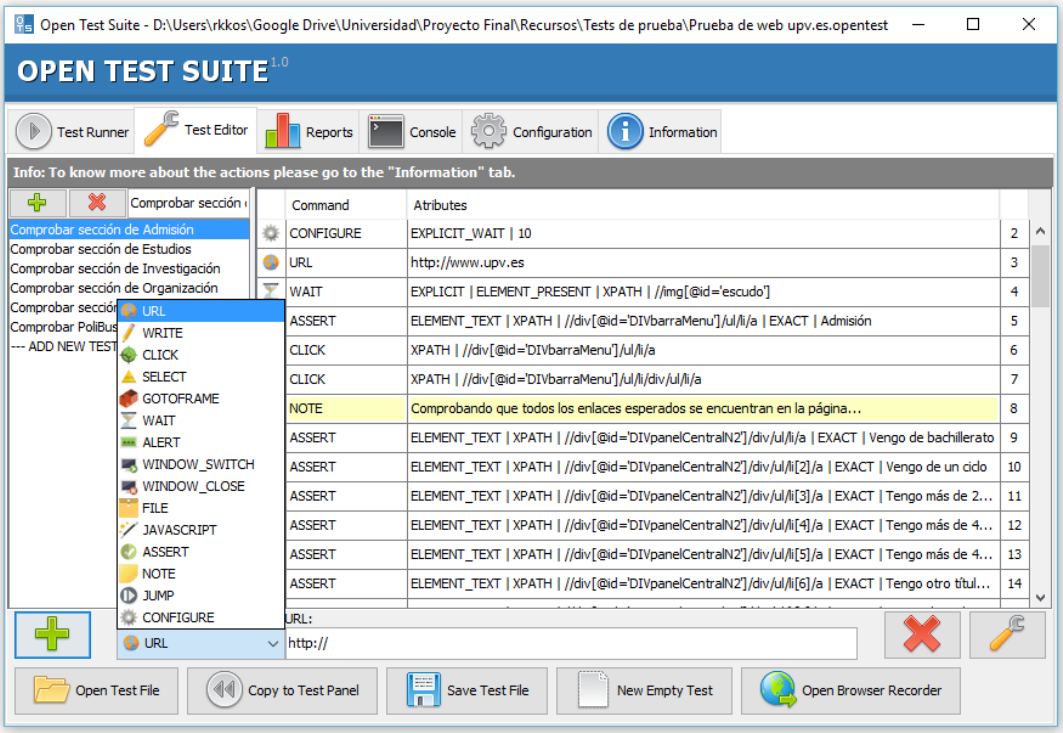
5.3.3. Test Runner

La interfaz de Test Runner mantiene el mismo diseño que en el prototipo, sin embargo, se han añadido mejoras de usabilidad como que ahora se muestran los errores en una columna separada y muestra la acción con error (Antes se mostraba un *stack trace* que no era demasiado intuitivo. Además, si se hace doble clic sobre un error, automáticamente abre el archivo de test en el editor de tests y muestra la línea en el test que causó el error.

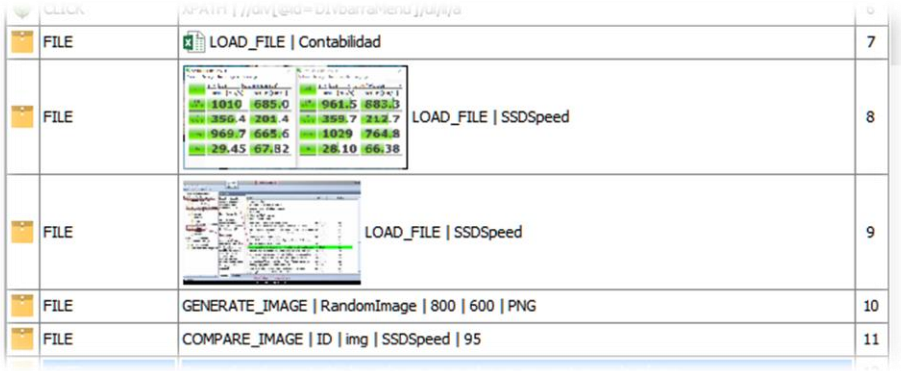


5.3.4. Editor de Tests

El editor de tests es el que probablemente ha recibido un mayor cambio interno.



Se ha añadido la posibilidad de añadir ficheros binarios a los tests, como también la posibilidad de generar imágenes aleatorias y con diferentes configuraciones en tiempo real durante las pruebas.



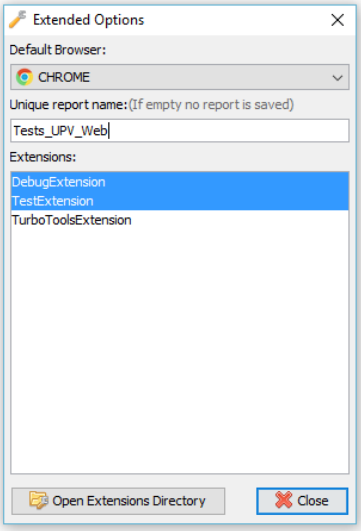
Los archivos binarios o imágenes se pueden usar durante la prueba para hacer pruebas de integridad de los archivos, permitiendo subirlos al servidor, como también descargar un archivo del servidor y compararlos binariamente y ver si coinciden.

Para imágenes, además de la comparación binaria, se puede hacer una comparación visual ingenua, en el que se compara la similitud visual de los pixeles, en el que se obtiene un porcentaje del 0 al 100 de lo que se asemejan la imagen del servidor con la que se tiene en el test. Esto es muy útil ya que normalmente en los servidores se suelen comprimir y optimizar las imágenes que se reciben y una comparación binaria fallaría inmediatamente.

Las acciones (**Actions**) como la de navegar por *Frames* y *IFrames* ha sido simplificada enormemente y acepta la navegación rápida por multiples frames sin necesidad de crear **Actions** por cada uno de ellos.

Se han añadido nuevas Actions que permiten simular una interacción a velocidad humana (Por defecto las **Actions** se ejecutan lo más rápido posible), esto puede ser muy útil en algunos casos.

El menú de **Extended Options** se ha mejorado ligeramente, siendo visualmente más atractivo e intuitivo.

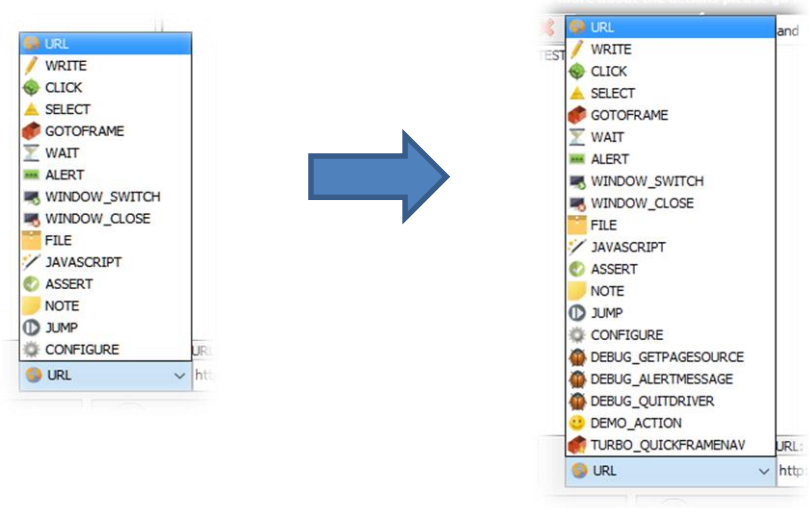


Se ha optimizado ligeramente la API de las Actions para las Extensiones, para que soporten los nuevos comandos de Wait.

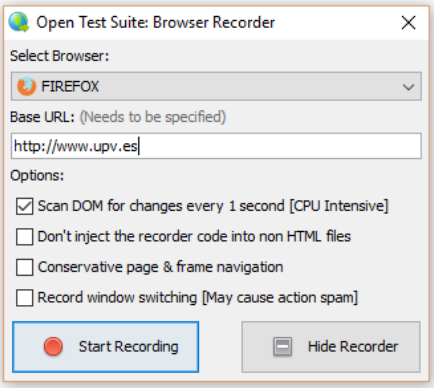
Se han añadido códigos de ejemplos para las extensiones, para que usuarios más avanzados puedan programar nuevas acciones para Open Test Suite sin necesidad de modificar todo el programa.

Las extensiones tras ser habilitadas son cargadas y añadidas automáticamente a la lista de acciones disponibles en el editor.

Sin embargo, nueva función más importante añadida es el "**Browser Recorder**", el cual se detallará a continuación.



5.3.5. Browser Recorder



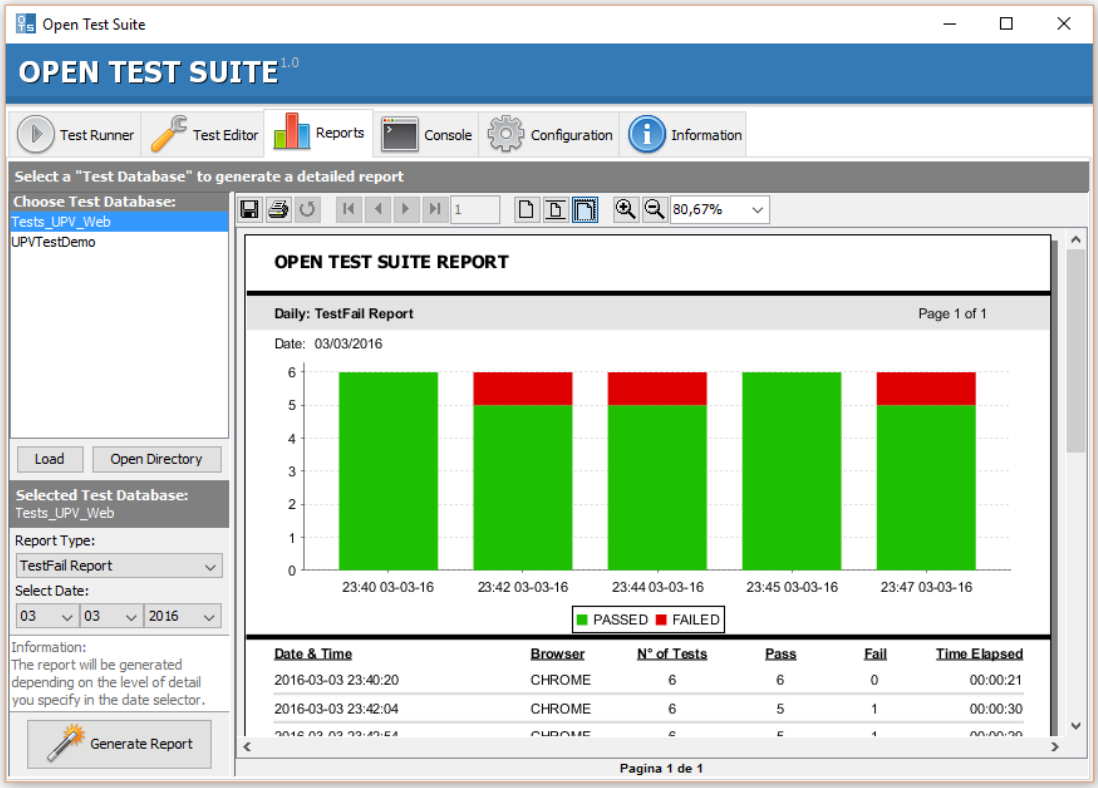
El *Browser Recorder* abre una página web en el navegador seleccionado y permite grabar en tiempo real lo que el usuario hace en él. La solución es completamente independiente de la plataforma y funciona en todos los navegadores que se puedan usar en **Open Test Suite**.

A parte de simplemente navegar por la página y ser grabado automáticamente, se puede grabar en directo sobre el editor de tests, permitiendo eliminar, cambiar o añadir acciones en el editor y la grabación continua dónde se tenga el cursor en ese momento, estando en todo momento consciente del contexto en el que se encuentra y actuando según sea necesario en dicho contexto (Muy útil en interfaces con muchos iframes).

Durante la grabación se puede ver el control de la grabación en todo momento, como también el último comando registrado. Cuando se presiona CTRL+ALT y se hace clic sobre algún elemento de la web, se abrirá el menú avanzado de opciones en el que se podrá realizar acciones como ASSERT, WAIT y FILE, y en el que los datos quedan rellenados automáticamente del elemento seleccionado en el navegador.

5.3.6. Reports

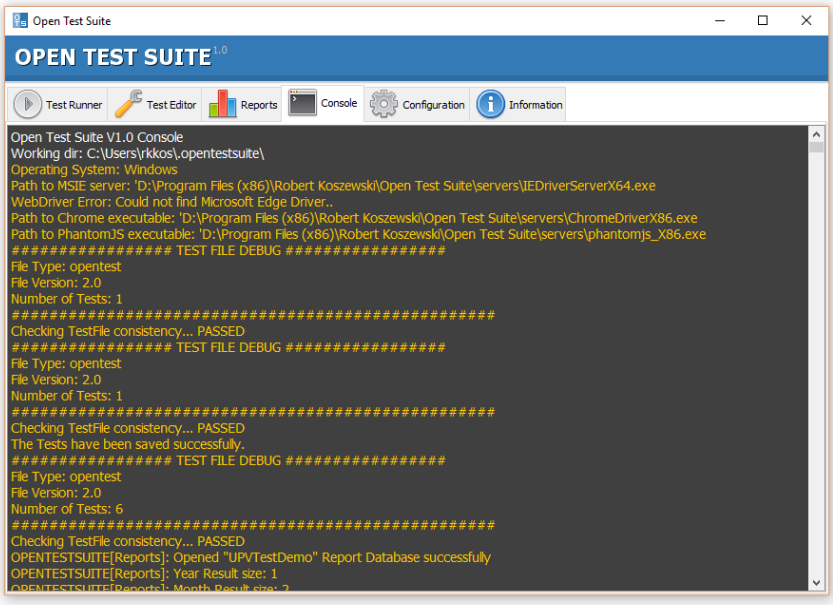
Ésta sección no ha cambiado demasiado desde el Prototipo. Se han mejorado levemente el diseño de los informes, destacando los datos más importantes.



Se ha hecho que la letra de la lista cronológica sea más pequeña, y los gráficos tienen colores más acordes con los gráficos que representan.

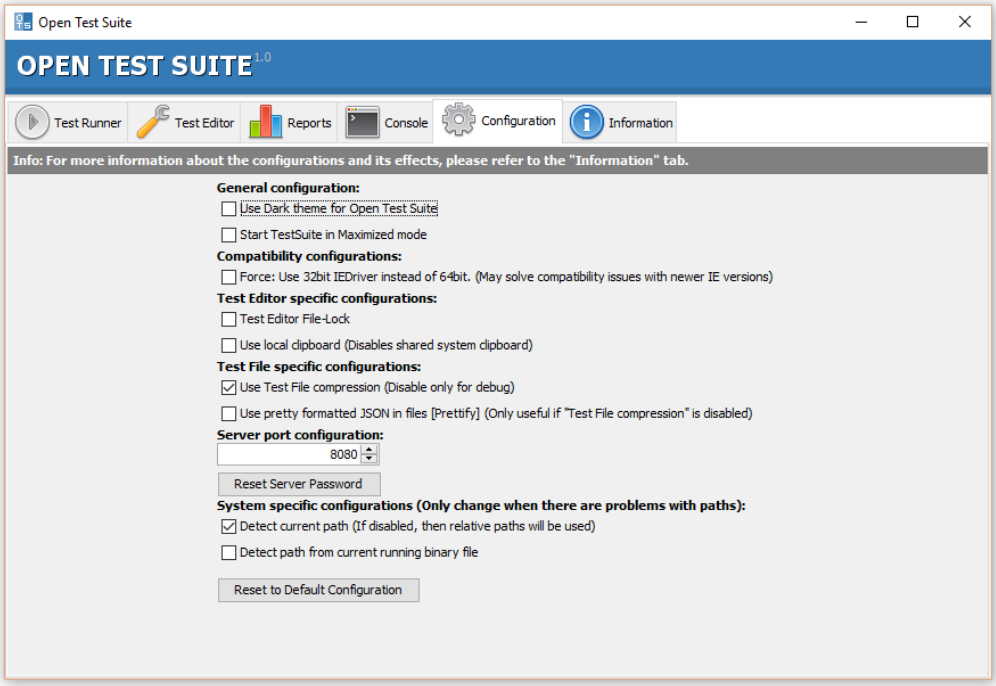
5.3.7. Console

La sección de **Console** apenas ha cambiado. Se ha limitado el buffer de texto para que consuma menos memoria RAM.



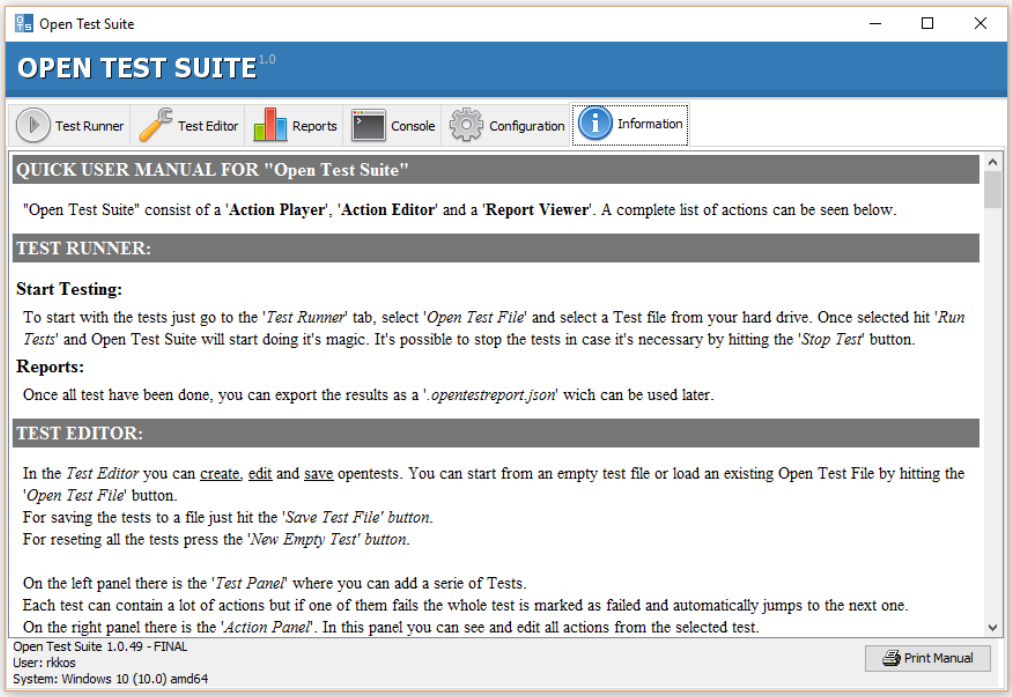
5.3.8. Configuration

En éste apartado simplemente se han añadido las nuevas opciones como la activación del “Modo Oscuro” y opciones de compatibilidad como usar el IEDriver de 32bits.



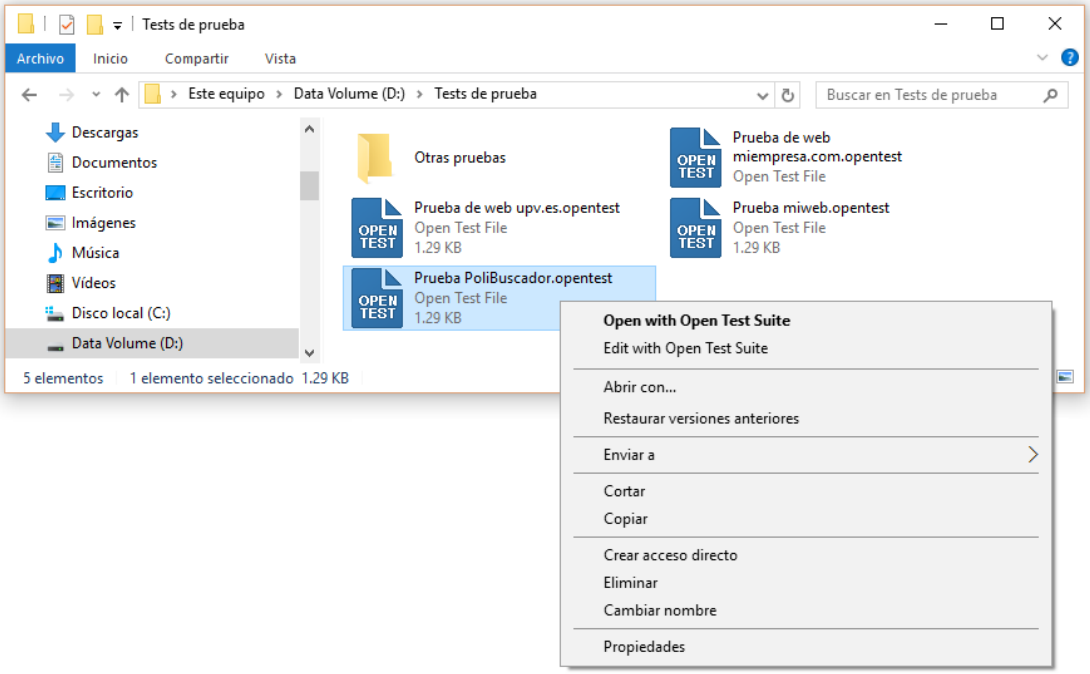
5.3.9. Information

El manual ahora es mucho más visual y con iconos. En vez de ser en puro ASCII ahora se usa HTML para el texto. Además, ahora se permite imprimir todo el manual si así se desea.



5.3.10. Integración con el sistema operativo

La versión se integra en el sistema operativo (De momento solo en Windows), muestra el archivo *.opentest* con su icono, y si se presiona con el botón derecho sobre el archivo, permite abrirlo en el "Test Runner" o en el "Test Editor" según se deseé.

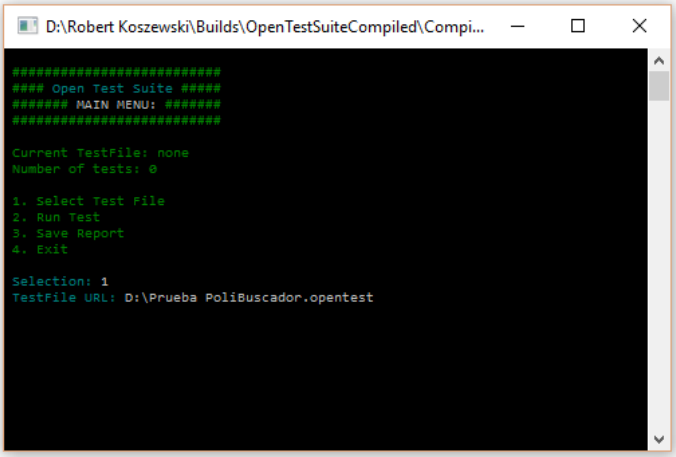


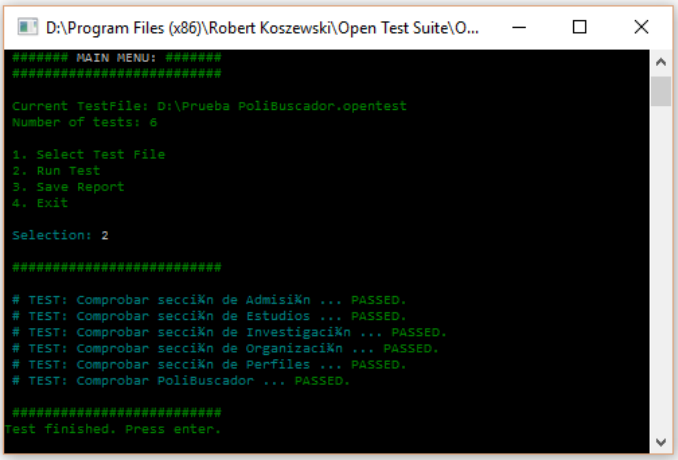
5.3.11. Console Modo

El modo consola (**Console Mode**) tampoco ha sufrido muchos cambios. Unos pocos cambios estéticos y solución de un par de errores son todo lo que ha cambiado desde el prototipo.

Primero hay que especificarle la dirección al archivo *.opentest* y se puede proceder con la ejecución de los tests.

Una vez cargado el test se puede seguir con su ejecución.





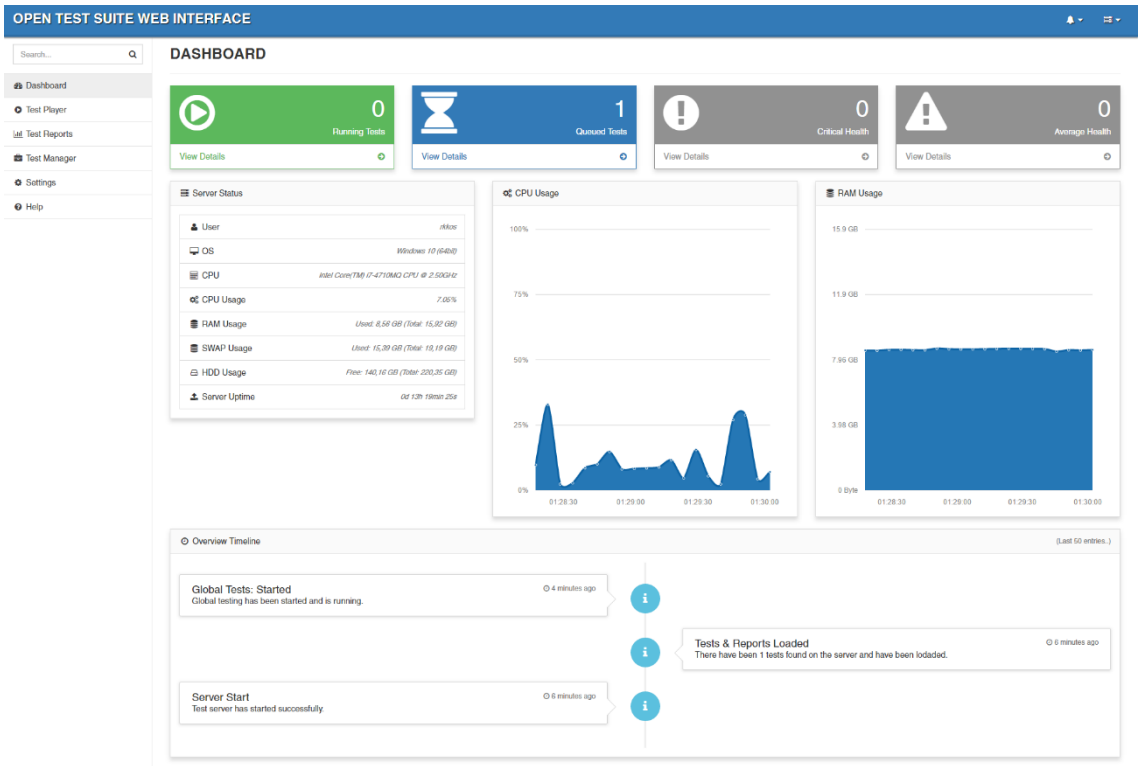
Se puede seguir el progreso del test directamente desde la consola y una vez finalizado, se puede guardar el resultado en JSON para su uso posterior.

5.3.12. Modo Servidor (Web) - Introducción

Éste es el apartado que ha sufrido el mayor cambio desde el prototipo. Se ha diseñado una interfaz que permita programar y gestionar un servidor dedicado a los tests desde una interfaz web. La interfaz del prototipo solamente permitía ejecutar un único test y con una interfaz muy limitada. En la versión final es mucho más completa, da soporte para que varios usuarios puedan trabajar sobre el mismo servidor al mismo tiempo. Todo se detallará a continuación.

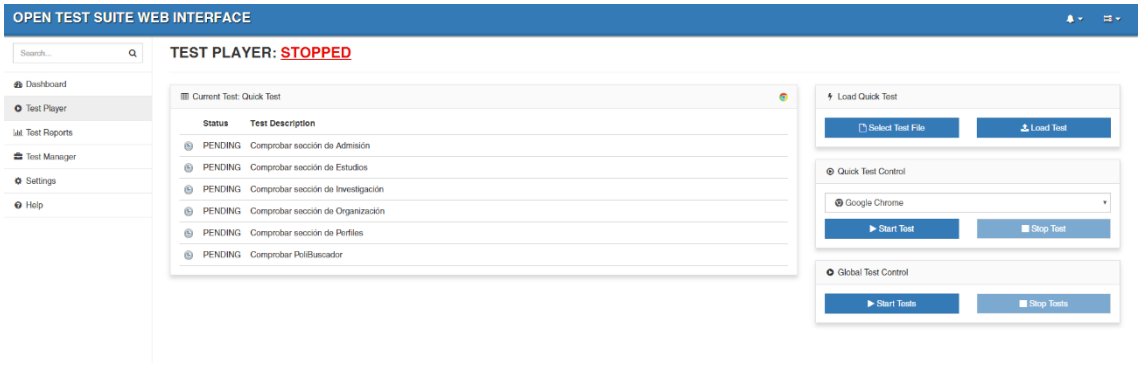
5.3.13. Modo Servidor (Web) – Página principal

En la página principal se puede obtener un vistazo general de todo lo que pasa en el servidor. Los tests ejecutándose, estado de memoria y CPU del servidor y un log de los últimos acontecimientos ocurridos en el servidor.



5.3.14. Modo Servidor (Web) – Test Player

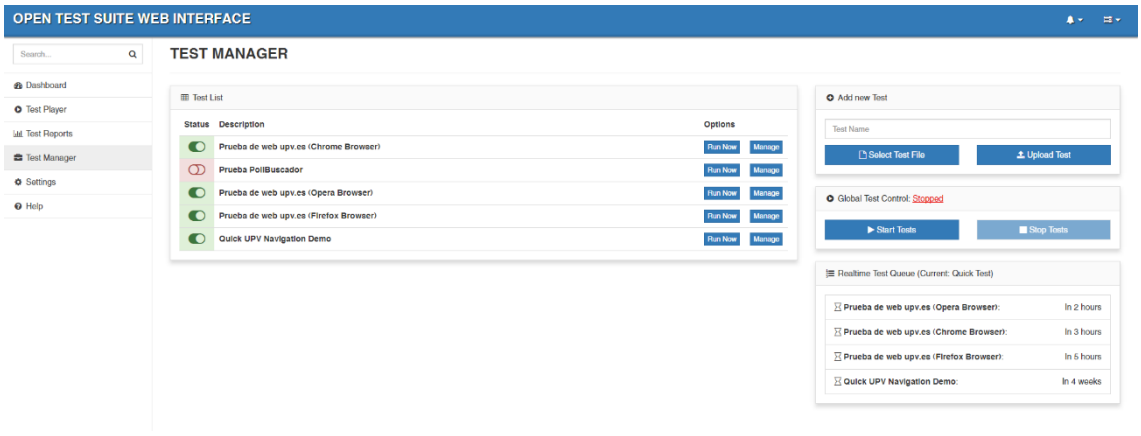
Dese el Test Player se pueden ejecutar test unitarios, llamados “Quick Test”, que es básicamente subir y ejecutar un único test que no se almacenará en el servidor.



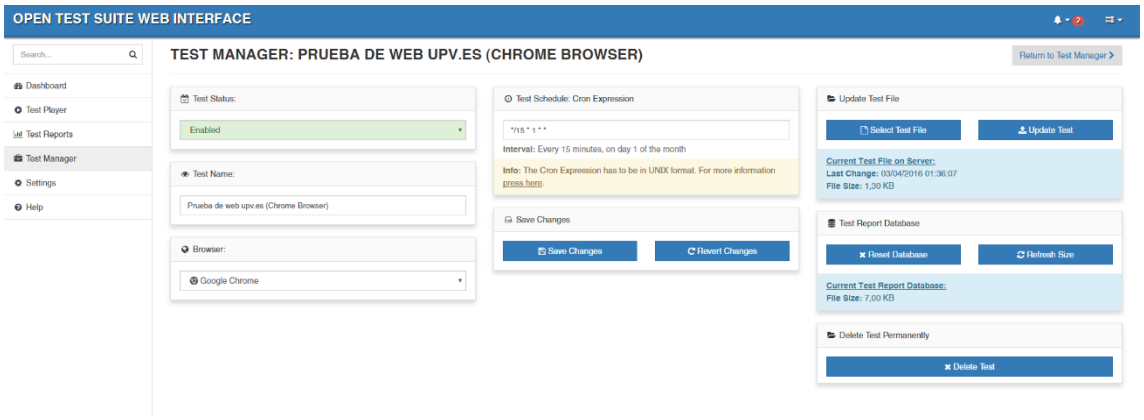
Como se puede ver en la foto, hay dos tipos de controles. Uno es el “**Quick Test Control**” y el otro el “**Global Test Control**”. El “**Quick Test Control**” permite controlar el test único subido desde el **Test Player**. Por otro lado, el “Global Test Control” controla la ejecución de los tests globales, que son los tests almacenados en el servidor. Ambos controles son excluyentes entre sí, no pueden ejecutarse ambos al mismo tiempo.

5.3.15. Modo Servidor (Web) – Test Manager

En éste apartado se pueden gestionar los tests globales, que son almacenados en el servidor. Éstos tests son básicamente archivos *.opentest* que se suben al servidor desde éste apartado y al que le es asignado un nombre único. Una vez subido en el servidor se pueden programar, habilitar/deshabilitar, gestionar y ejecutar directamente en el “**Test Player**”.



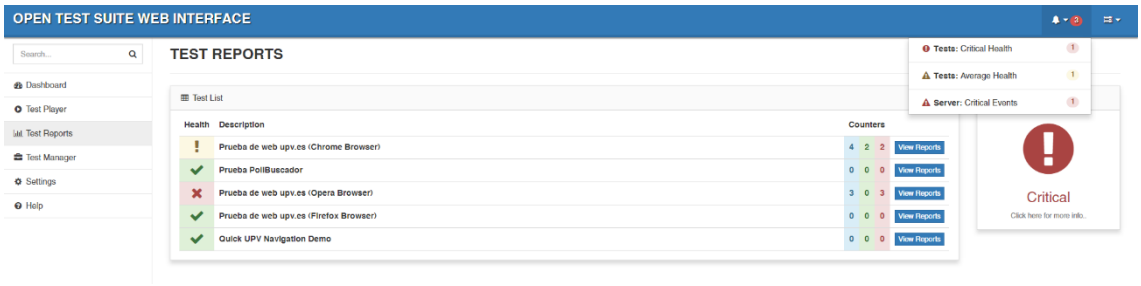
Cuando un test es subido al servidor en ésta sección, se puede gestionar presionando sobre el botón “**Manage**”.



En ésta sección se puede gestionar cada test. Entre otras cosas se permite cambiar el navegador porque se usará para éste test, si se quiere especificar uno diferente al por defecto especificado en el test. Se permite actualizar el test para que no se pierdan las estadísticas. Y lo más importante es la programación del test. Éstos se pueden programar en formato **UNIX** para ser ejecutado cada cierto tiempo o a una fecha específica. Un breve tutorial sobre la sintaxis se puede ver al presionar el botón de ver más información.

5.3.16. Modo Servidor (Web) – Test Reports

En ésta sección se pueden ver las estadísticas de las ejecuciones de los tests globales como también acontecimientos críticos del servidor. Como se puede ver en la imagen, desde el desplegable de arriba se pueden consultar en todo momento los acontecimientos más importantes que llevarán a éste apartado.

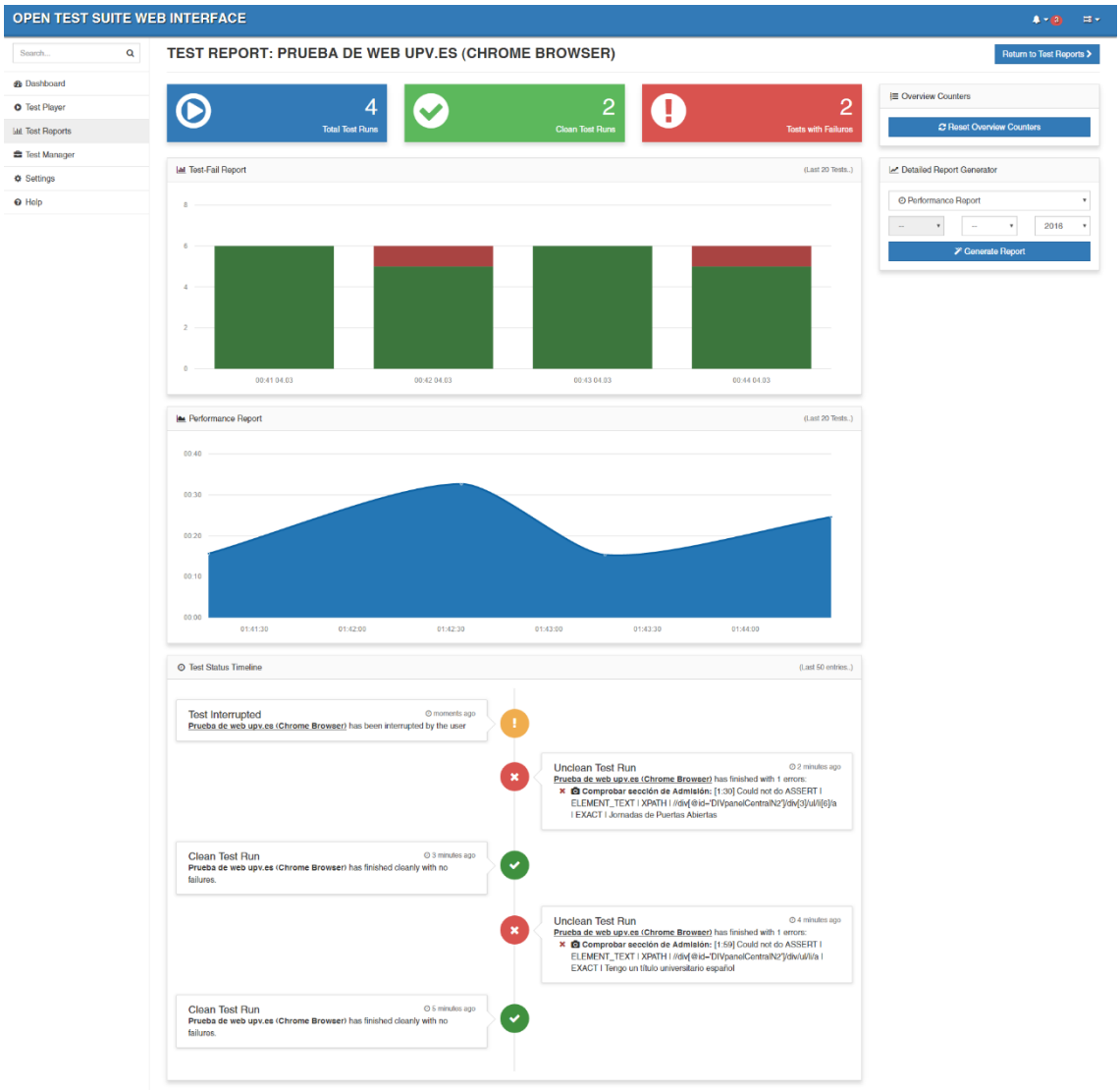


Los tests que nunca han tenido problemas son marcados como saludables, los que sí han tenido errores, pero debajo del umbral crítico, se ven como "cuidado". Los que superan el umbral crítico se muestran como críticos (El umbral crítico se puede cambiar desde el apartado "**Settings**" que será descrito posteriormente). La decisión de usar un umbral configurable de error se debe a que a veces los errores pueden ser causados por motivos no relacionados con la aplicación web como pérdida de conexión, el navegador no responde, etc.

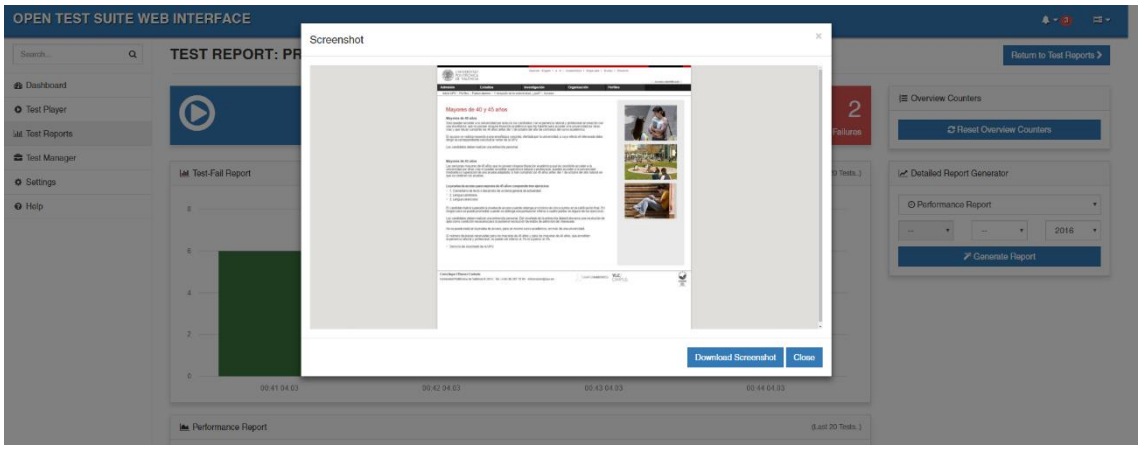
Desde aquí se puede acceder a los Reports detallados de cada test haciendo clic sobre "View Reports".

5.3.17. Modo Servidor (Web) – Reports Detallados

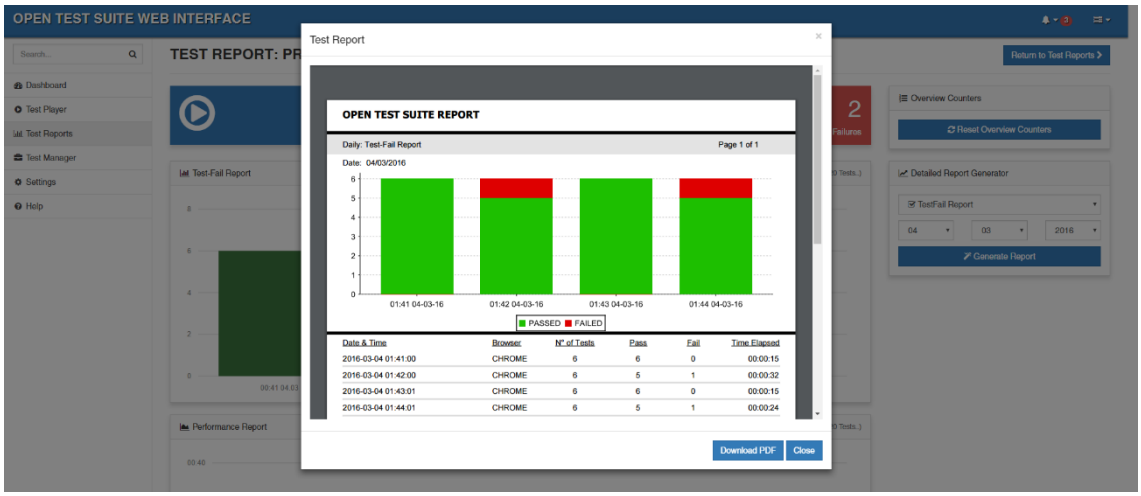
En éste apartado se pueden consultar las estadísticas, generar informes y ver los últimos acontecimientos de éste test.



Desde aquí es dónde se pueden reiniciar los contadores de errores, para cuando el problema ya ha sido resuelto. Desde aquí también se puede segur el estado de los últimos tests, y en caso de tests con fallos, ver una captura de pantalla de los tests con errores, para poder ver más fácilmente el momento en el que el test falló.



Desde aquí también se pueden generar informes detallados, que tienen la misma estructura como los que se pueden ver en el apartado "**Reports**" del modo GUI de la interfaz local del programa.

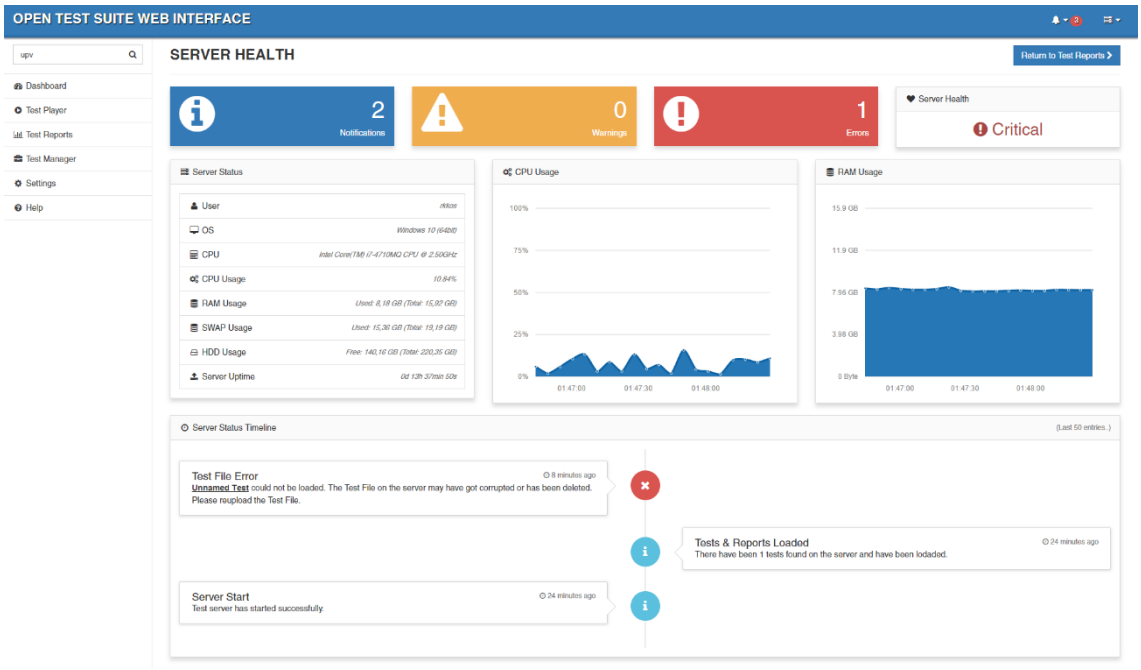


5.3.18. Modo Servidor (Web) – Server Health

Se puede acceder a éste apartado desde la sección de "**Test Reports**". Aquí se pueden ver los últimos acontecimientos relacionados con el servidor, y ver su estado.

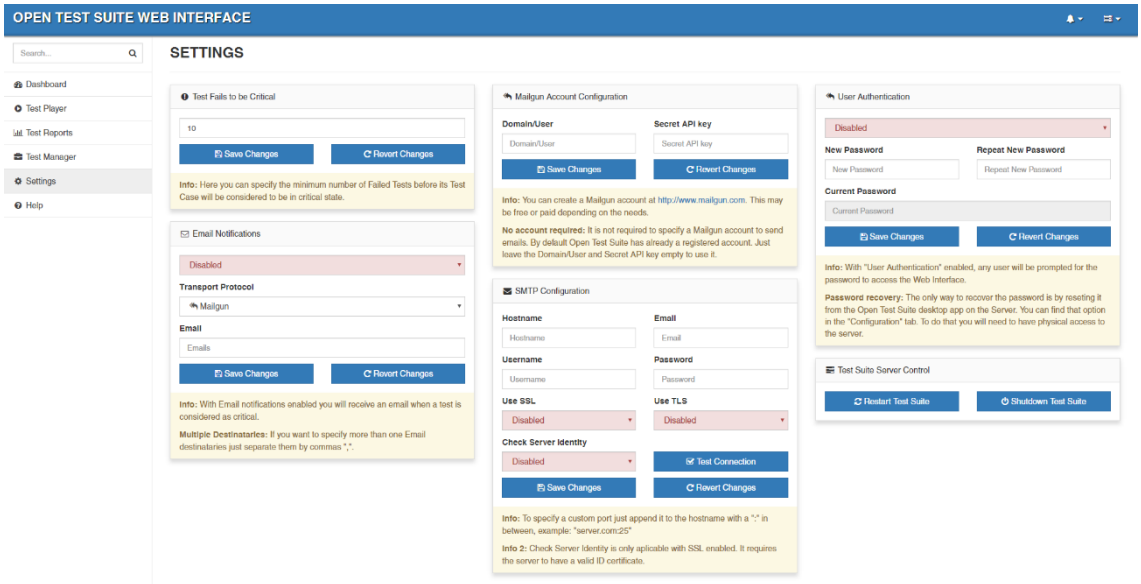
También se muestra una gráfica del uso del procesador y uso de memoria RAM. En el caso de que el uso de CPU permanezca en 100% durante más de 10 minutos, se mostrará un mensaje de cuidado, al igual si el uso de la RAM permanece por encima del 95% durante el mismo periodo de tiempo.

El uso del disco duro también se monitoriza en todo momento, y genera un mensaje de cuidado si se dispone menos de 250MB de disco duro, lo que podría causar problemas de espacio y prevenir el correcto funcionamiento de los tests.



5.3.19. Modo Servidor (Web) – Settings

En éste apartado se puede acceder a la configuración del servidor. Desde aquí se puede establecer el umbral de error de los tests globales, configurar notificaciones email que se enviarán cuando el umbral de error se rebase, para que no sea necesario estar pendiente del servidor, sino que el servidor notifique al usuario cuando se rebase el umbral. Se soporta envido por *SMTP* como también usando *Mailgun*. Cada opción es explicada con detalle en el marco amarillo debajo de cada opción, permitiendo que incluso usuarios inexpertos puedan empezar a usar la aplicación con relativa sencillez.

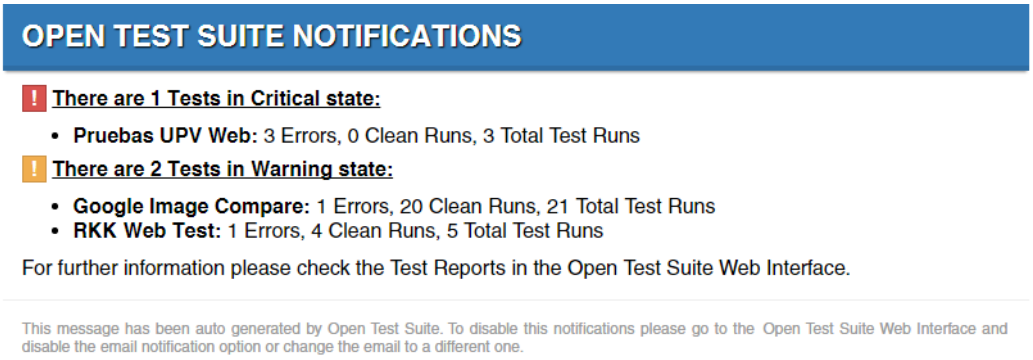


Desde aquí se puede establecer una contraseña para la interfaz web, para restringir el acceso a la interfaz por parte de personas no autorizadas.

Finalmente, desde aquí se puede apagar o reiniciar el servidor si fuera necesario (Se reinicia el software, no el Sistema Operativo).

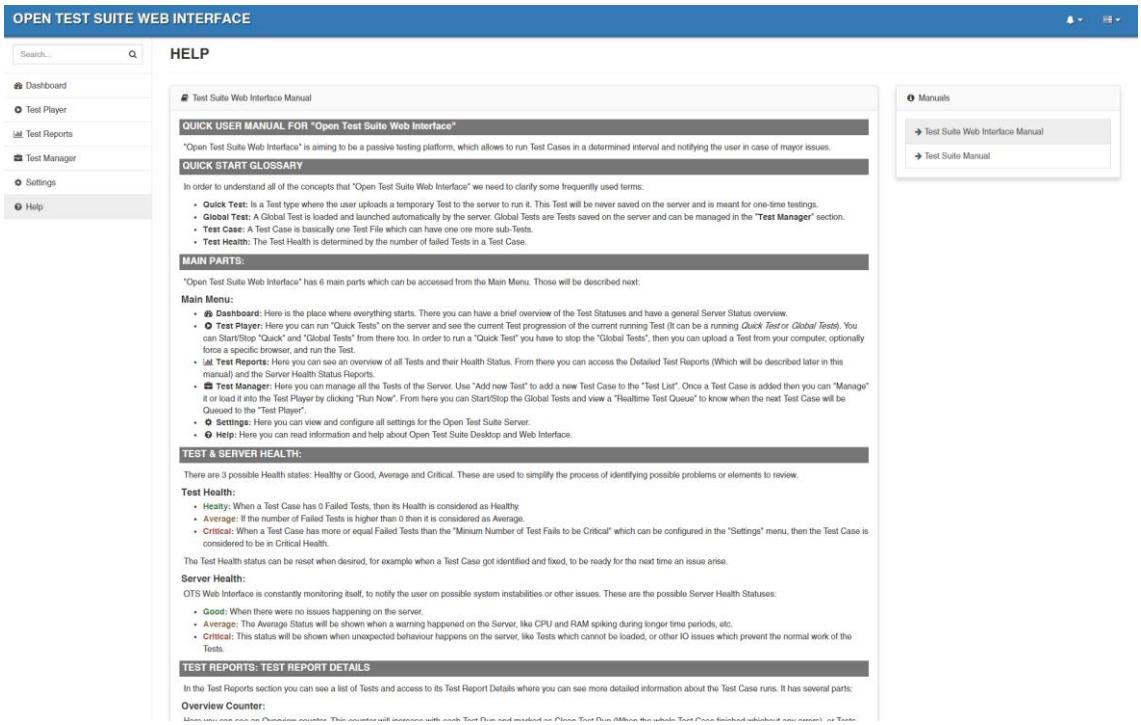
5.3.20. Modo Servidor (Web) – Notificaciones email

Las notificaciones se envían a los destinatarios configurados en el apartado "**Settings**" cuando algún test global rebasa el umbral crítico. El email tiene la siguiente estructura:



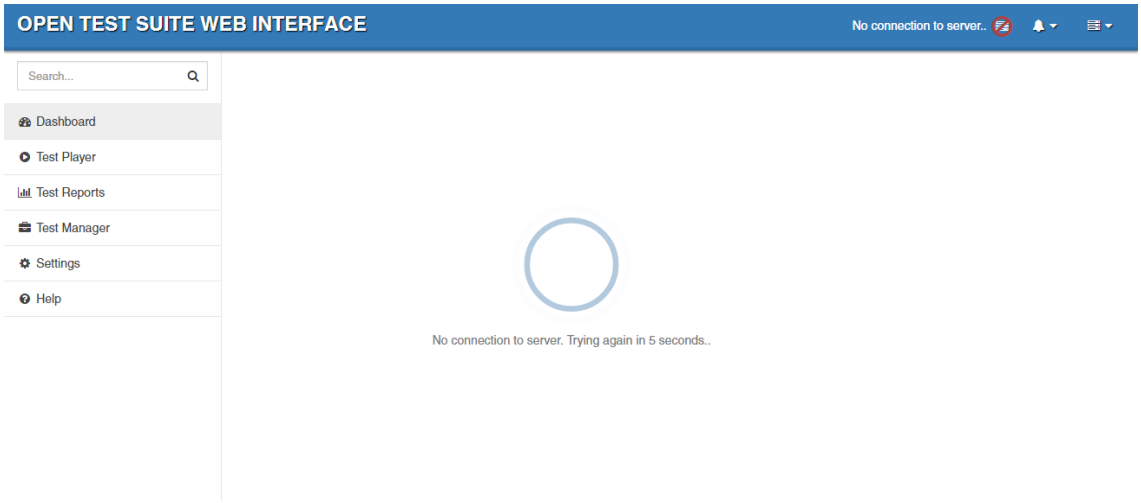
5.3.21. Modo Servidor (Web) – Help

En éste apartado se puede obtener ayuda e información sobre el uso del servidor, como también acceder al manual de Open Test Suite.

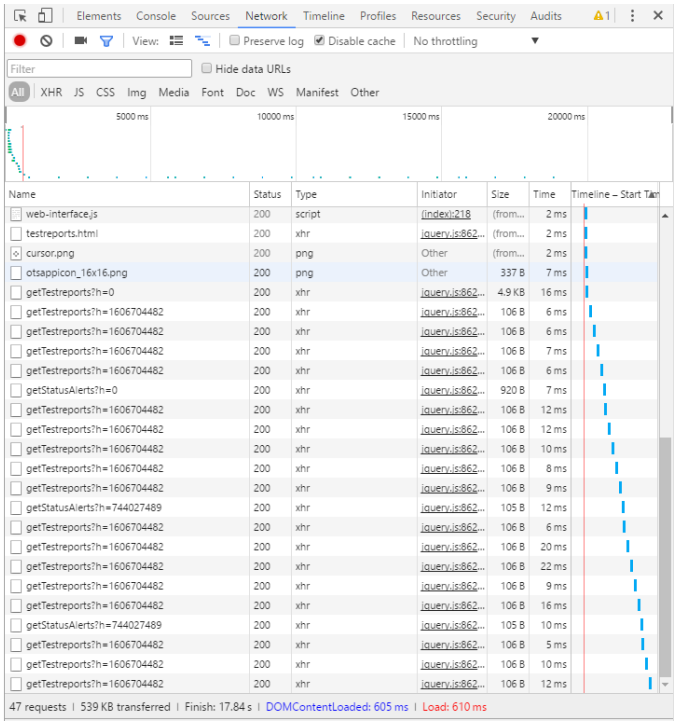


5.3.22. Modo Servidor (Web) – Funcionamiento offline y móvil

La interfaz del servidor está diseñada para ajustarse a todo tipo de pantallas. Desde móviles hasta pantallas de alta resolución (Se usa Bootstrap). La aplicación funciona en modo offline, y cuando pierde la conexión con el servidor sigue funcionando esperando la respuesta del servidor, y notificando de ello al usuario. Dada su naturaleza offline, la interfaz se carga solo una vez y nunca más. El resto de datos se almacenan en una memoria HTML 5, que evita descargar componentes que se han descargado previamente, incluso si se ha eliminado la caché del navegador.



La comunicación con el servidor es en tiempo real, con retardo máximo de 1s para las acciones críticas (frecuencia de actualización de 1s). Sin embargo, la conexión ha sido optimizada para que se envíen lo menos datos como sea posible. Enviando datos al cliente tan solo cuando han cambiado.



Como se puede ver la primera llamada a los datos pesaba 4.9KB, mientras que todas las llamadas siguientes tan solo pesan 106B.

Esto ahorra en gran medida datos cuando se realiza una conexión desde una conexión medida de internet o móvil.

6. Conclusiones y líneas futuras de desarrollo

El programa final ha resultado bastante útil y ya está siendo aprovechado actualmente al menos por un equipo de testing (QA) con lo que se puede decir que su usabilidad ha quedado demostrada.

6.1. Estadísticas del código fuente

Las estadísticas son del código fuente de la aplicación principal, no se cuentan los archivos de los diseños, código de los instaladores, servidores y recursos externos.

Lenguaje	Archivos	Lin. Comment.	Lin. Código
Java	119	3927	14587
CSS	8	78	3702
HTML	15	99	1979
JavaScript	15	189	1028
XML	1	26	36
TOTAL:	**158**	**4319**	**21332**

6.2. Futuro del desarrollo de la aplicación

Aunque el programa es totalmente funcional, ha sido diseñado principalmente para ser usado en entornos monousuario, y la implementación del servidor para acceso multiusuario y su modo de funcionamiento es una solución temporal que simula la posibilidad de ejecutar varios tests sobre un motor monousuario.

Para el futuro sería interesante usar tecnologías más modulares como OSGi que permiten romper el programa en servicios y receptores, permitiendo una implementación que puede ser libremente, actualizada y que sea lo suficientemente flexible como para permitir correr en entornos mono y multiusuario permitiendo realizar varios tests en paralelo si así se desea.

Otra adición para el futuro es añadir la posibilidad de que Open Test Suite se extienda a otros ámbitos de testing que no solo sea web.

7. Bibliografía

- Documentación de Selenium - **http://www.seleniumhq.org/docs**
- What is Regression Testing? - **https://smartbear.com/learn/automated-testing/what-is-regression-testing**
- Definitive guide to Swing for Java 2 - John Zurowski (1999)
- Basics of Java class loaders - **http://www.javaworld.com/article/2077260/learn-java/learn-java-the-basics-of-java-class-loaders.html**
- Selenium 2 Testing Tools: Beginner's Guide - David Burns (2012)
- Java Platform SE 7 Documentation (Oracle) - **https://docs.oracle.com/javase/7/docs/api/**
- StandardPrint Example - **https://community.oracle.com/thread/1289442**
- Netty 4 Documentation - **http://netty.io/4.0/api/**
- Netty 3 User Guide - **http://netty.io/3.8/guide/**
- Selenium Javascript DOM Selector - **http://stackoverflow.com/questions/11430773**
- Naïve Similarity Finder Algorithm (Image Comparison) - **http://www.lac.inpe.br/JIPCookbook/6050-howto-compareimages.jsp**
- Random Image Generation Algorithm - **https://www.dyclassroom.com/image-processing-project/how-to-create-a-random-pixel-image-in-java**
- Selenium File Downloader - **http://sqa.stackexchange.com/a/2595**
- Selenium Chrome Driver - **https://sites.google.com/a/chromium.org/chromedriver/**
- PhantomJS - **http://phantomjs.org/**
- FatCow Free Icon Pack - **http://www.fatcow.com/free-icons**
- SB Admin 2 Bootstrap Template - **http://startbootstrap.com/template-overviews/sb-admin-2/**
- Apache IVY - **http://ant.apache.org/ivy/history/2.0.0/tutorial.html**
- Apache Mime4J - **https://james.apache.org/mime4j/**
- Apache Commons - **https://commons.apache.org/**
- Commons Email - **https://commons.apache.org/proper/commons-email/**
- Mailgun API - **https://documentation.mailgun.com/wrappers.html**

- Cron4J - **http://www.sauronsoftware.it/projects/cron4j/**
- Cron Utils - **https://github.com/jmrozanec/cron-utils**
- Google GSON - **https://github.com/google/gson**
- HtmlUnit - **http://htmlunit.sourceforge.net/**
- ImgScalr - **http://www.thebuzzmedia.com/software/imgscalr-java-image-scaling-library/**
- Ini4J - **http://ini4j.sourceforge.net/**
- Jasper Reports Library and Tools - **http://community.jaspersoft.com/project/jasperreports-library**
- JBusyComponent - **https://code.google.com/archive/p/jbusycomponent/**
- JMimeMagick - **https://github.com/arimus/jmimemagic**
- Joda Time - **http://www.joda.org/joda-time/**
- Introduction to JSON and Libraries - **http://www.json.org/**
- JSOUP (Java HTML Parser) - **http://jsoup.org/**
- JZLib (Zlip in pure Java) - **http://www.jcraft.com/jzlib/**
- Sigar Library - **https://support.hyperic.com/display/SIGAR/Home**
- Pretty Time - **http://www.ocpsoft.org/prettytime/**
- SQLite in Java - **http://www.tutorialspoint.com/sqlite/sqlite_java.htm**
- SwingX - **https://java.net/projects/swingx**
- WebLAF - **https://github.com/mgarin/weblaf**
- Jansi - **https://github.com/fusesource/jansi**
- Java JACOB (Java COM bridge) - **https://sourceforge.net/projects/jacob-project/**
- Ant Tutorial - **https://ant.apache.org/manual/tutorial-writing-tasks.html**
- Launch4j - **http://launch4j.sourceforge.net/**
- NSIS - **http://nsis.sourceforge.net/**
- Inno Setup - **http://www.jrsoftware.org/isinfo.php**
- Fix IE11 Issues with Selenium - **https://github.com/seleniumhq/selenium-google-code-issue-archive/issues/6511**
- LittleProxy (Java Proxy w. MITM capability) - **https://github.com/adamfisk/LittleProxy**

8. Anexos

9.1. Manual de Open Test Suite

QUICK USER MANUAL FOR "Open Test Suite"

"Open Test Suite" consist of a '**Action Player**', '**Action Editor**' and a '**Report Viewer**'. A complete list of actions can be seen below.

TEST RUNNER:

Start Testing:

To start with the tests just go to the '*Test Runner*' tab, select '*Open Test File*' and select a Test file from your hard drive. Once selected hit '*Run Tests*' and Open Test Suite will start doing its magic. It's possible to stop the tests in case it's necessary by hitting the '*Stop Test*' button.

Reports:

Once all test have been done, you can export the results as a '*.opentestreport.json*' wich can be used later.

TEST EDITOR:

In the *Test Editor* you can create, edit and save opentests. You can start from an empty test file or load an existing Open Test File by hitting the '*Open Test File*' button.
For saving the tests to a file just hit the '*Save Test File' button*.
For reseting all the tests press the '*New Empty Test' button*.

On the left panel there is the '*Test Panel*' where you can add a serie of Tests.
Each test can contain a lot of actions but if one of them fails the whole test is marked as failed and automatically jumps to the next one.
On the right panel there is the '*Action Panel*'. In this panel you can see and edit all actions from the selected test.

To add a new element just press on the '*ADD NEW TEST*' space or in the '*Actions Panel*' in the white space from the bottom.
If you select an existing element and you press the ADD button it will modify the existing element and won't add a new one.
You can remove any elements at any time.

TEST EDITOR: Action Lexicon

Here is a list of all available actions that can be performed:

Navigation and Basic Interaction:

- **URL:** Navigates to the selected URL.
- **WRITE:** Writes text into a specified element.
- **CLICK:** Clicks on a specified element.
- **SELECT:** Used for 'Drop-Down' elements.

- **GOTOFRAME:** Allows navigating through Frames and iFrames in the DOM. You can choose between different modes:
  - **DEFAULT FRAME:** Selects the root frame of the document.
  - **FRAME NAME:** Selects the frame by its name.
  - **FRAME INDEX:** Selects the frame by its index between all frames in the document.
  - **FRAME PATH RELATIVE:** You can specify different frames separated by a ">" this command will then navigate trough all of them. If you want to use the index of the frame you have to write the index number between "|". Examble: "|*1*|".
  - **FRAME PATH ABSOLUTE:** This command works exactly like the previous one just that it starts from the root document frame.
- **WAIT:** Waits for the specified amount of milliseconds or for a specified element or condition.
- **ALERT:** Allows interacting with browser ALERTs, PROMPTs, AUTHs and CONFIRMs.
- **WINDOW_SWITCH:** Allows switching between windows when more than one is used for browsing.
- **WINDOW_CLOSE:** Closes the current selected window.
- **FILE:** Allows to interact with File related elements
  - **LOAD FILE:** Allows to add a file to be used during the Test. You can specify a unique internal **File ID** to select the file during the test (By default it uses the file name as File ID). **Please note:** If you wish to assert the *file name* during the test you have to use the original name of the file (Not the internal File ID). You can see the original filename and path by selecting the action and hovering on the icon next to the "Open File Chooser" button.
  - **SET FILE:** Sets a file on a *INPUT:FILE* element inside a **form**. This is the equivalent of opening the **file selection dialogue** in the browser and select a file.
  - **COMPARE FILE:** This allows to download and compare a file with an other included in the Test. **Please note:** It is discouraged to use this method during tests if possible as Selenium doesn't supports this feature directly and as such may fail in some cases. It can only download the file if the file is specified in an element in the "**href**" tag (It won't work if the link to the download is specified via Javascript or other). If authentification is required to download the file and "HTTPOnly" cookies are being used by the server, the download will fail too.
  - **COMPARE IMAGE:** This allows to compare a web element with an image. It is mainly used to check if an image has been uploaded or to confirm that an image is similar to the expected one.
    Please note (1): Selenium takes a screenshot of the selected element, which means that if it has some kind of overlay or is ocluded by something, the overlapped content will be visible too, so it's recommended to lower the similarity percentage. As it is a screenshot of an element it can be used to compare the correct design of static objects by comparing it with an other screenshot made previously for the test.
    Please note (2): The images on the server may differ slightly from the original

because of compression artifacts or other transformations made on the server side so an 100% of similarity is not very usual. It is recommended to use a value of 95 to 99% of similarity for clear non-overlapped images.
Please note (3): This is a pure perceptive comparison, and as such the results may not be 100% accurate or may even confuse similar looking images.
    - **GENERATE IMAGE:** Generates a random generated image, optimized for image comparison.
- **JAVASCRIPT:** Allows executing of Javascript code in the browser.
- **ASSERT:** Checks if a specified condition is met (If not, then it throws an error)
- **NOTE:** Used to write Notes in the Test Editor.
- **JUMP:** Allows conditional jumping inside the test.
Please note that the state EXCEPTION can only be accessed if the option CONFIGURE->FAIL ON EXCEPTION is OFF.
Don't forget to use CONFIGURE->CLEAR EXCEPTION STATE if you want to clear previous exception states.
WARNING! This function can cause INFINITE LOOPS if not used correctly.
- **CONFIGURE:** Allows to configure Selenium in realtime.
    - **SET VARIABLE:** Allows setting variables which can be used later in the test. Variables only work if ENABLE VARIABLES is "ON" (By Default it is "ON"). You can use them by writing "*${variablename}*" into any of the Actions inputs and will be replaced by the specified value if the value is set.
    - **ENABLE VARIABLES:** Enables variables parsing. When enabled then you can set variables and use them later in all text elements by writing "*${variablename}*". By default this is set to "ON". Set this to "OFF" to avoid problems with ${} if required.
    - **IMPLICIT WAIT:** This is the number of seconds which Selenium will wait till an element is present. By default its "0" (Expected immediately).
    - **EXPLICIT WAIT:** This is the number of seconds which Selenium will wait till an element is present in "WAIT>EXPLICIT WAIT" mode. By default this is set to "10".
    - **POST ACTION WAIT:** Allows setting a amount of time to wait after each action is executed (Some Actions may skip this).
    - **IMPLICIT WAIT FRAME ELEMENT:** Allows setting a element which will be checked when a frame is changed. (It has the same effect like invoking "WAIT|EXPLICIT|ELEMENT PRESENT" after each frame change)
    - **TEST RETRIES:** Number of times a failed Test will be done again without giving an error. By default it is set to "0" (No retries).
    - **FAIL ON EXCEPTION:** If this option is set to "ON" then the Test will be given as failed if an exception occurs. (Default: "ON").
    - **AUTO CLEAR EXCEPTION STATE:** If enabled then every successful action overwrites the previous exception state (if any exists). This is only useful with FAIL_ON_EXCEPTION set to "OFF". If turned OFF you have to manually clear the exception state.

- **CLEAR EXCEPTION STATE:** Only useful if AUTO_CLEAR_EXCEPTION_STATE is set to "OFF". Clears the status of an exception setting it to successful.
- **STOP ON EXCEPTION:** If this option is set to "ON" when the Test fails, all subsequent Tests will be aborted. This is useful in cases when the subsequent Tests depend of the correct execution of the current Test (Example: With this setting set to "OFF", if the login process fails, all subsequent tests would fail even if they might be correct and working). **Please note:** All Test results till the aborted Test will be written to the Report if a Report name is specified. (Default: "OFF").
- **DELETE COOKIE:** Allows you to delete a specific cookie by its name.
- **DELETE ALL COOKIES:** Deletes all cookies.
- **WINDOW MAXIMIZE:** Maximizes the current selected window.

TEST EDITOR: Shortcuts

To remove multiple elements just press the **STRG** or **CAPS** button to select multiple elements and then just press the remove button.
Single and multiple selected elements can be copied, pasted, deleted and moved.

By pressing **ALT+UP** Key or **ALT+DOWN** key you can reorder the Tests or Actions.
Multiple selection while moving elements is also supported.
By pressing the **ALT** Key while having selected an existing Test or Action, will add an new element under the selected position instead of modifying it.
By pressing the **ALT** Key while clicking on "Save Test File" button. If a file has been selected or already saved once, then it will be saved directly.

You can use **CTRL+C** and **CTRL+V** to Copy and Paste Tests and Actions.
Press **CTRL+Z** or **CTRL+Y** to Redo or Undo the modifications made.

Save to New Test File Shortcut: By default the "Save Test File" button saves to the same Test File automatically. But when the ALT button is pressed and the button clicked, you can save the Test as a different file.

TEST EDITOR: Best Practices

When a *Test* fails, the failed *Action* will be displayed next to the Test. The error information starts with two numbers inside of brackets "[]" separated by a ":".
The first number represents the Test number where the Action failed and the second one the failed Action number ([test_number:action_number]).
By double clicking on the error in the Test Player, loads the Test in the Test Editor and highlights the failed Action automatically.

All other errors and debug information is shown in the "**Console**" tab..

REPORTS:

In the Reports section you can generate all kind of reports. Just select the report database, a report type, the date and press the generate report button. From there you can view, print or save the report into numerous formats.

Notes about dates: The report will be generated till the date selected. If only the year is selected then it will be a yearly report. If you select a year and a month then it will be a monthly report and so on.

MORE INFORMATION ON INTERNET AND TUTORIALS:

TUTORIAL: How to write XPATH Selectors:

http://hedleyproctor.com/2011/05/tutorial-writing-xpath-selectors-for-selenium-tests/

TUTORIAL: How to use CSS selectors:

https://saucelabs.com/resources/selenium/css-selectors

Documentation:

http://www.seleniumhq.org/docs/

"Open Test Suite" STARTUP OPTIONS:

Here you can see a list of available startup options for launching Open Test Suite.

- **-consolemode:** Starts Open Teststuite in "Console mode" allowing the control of the app from the console.
- **-runtest <URL to .opentest file>:** Starts the test automatically if a TestFile direction has been specified. If run together with "-consolemode" it will run the test.
- **-jsonout:** This command works only when -consolemode and -runtest is specified. Instead of showing a result in realtime, it will show a JSON report at the end of the test. (Useful for automated testing with report result parsing by 3rd party software).
- **-forcedebug:** Forces the debug output in "NoGUI mode".
- **-forceie32:** Forces "IE 32Bit Driver Mode".
- **-browser<BROWSERNAME>:** Forces the selected browser as default browser in the test if a testfile is specified and -runtest is enabled. You can find a list of supported browser in the next section "SUPPORTED BROWSERS". The browser has to be specified wichout spaces. Example "-browserFIREFOX" for Firefox or "-browserMSIE" Microsoft Internet Explorer.
- **-editormode:** Opens the "Test Editor" automatically.
- **-filelock:** Enable File Locking for the Test Editor. This ensures that only one user is editing a Test File at a time.
- **-nocheck:** Disables TestFile integrity checks. MAY PRODUCE UNPREDICTABLE RESULTS. DO NOT ENABLE THIS UNLESS IT'S STRICTLY NECESARY.
- **-maximizegui:** Starts the GUI in maximized state.
- **-localclipboard:** Disables the use of System Clipboard in Test Editor and uses a local one (Cannot share clipboard between app instances)
- **-explosions:** Shows some fancy explosions when a test fails (Good to know if something failed when the browser is in fullscreen mode)
- **-nocompress:** Disables compression while saving TestFiles so the files can be edited in a regular Text Editor.
- **-jsonpretify:** Enables JSON pretify while saving TestReportFiles and TestFiles.

- **-server:** Starts the server mode from wich you can run test from a Web interface by accessing the current ip with the selected port.
- **-setport<PORT NUMBER>:** Forces the use of the selected port for the server.
- You can specify the direction of a existing .opentest file to load it automatically in "Run Test" or if "-editormode" is set it will be automatically loaded into the "Test Editor".
- Most of the configurations can be set permanently in the "Configuration" tab.

SUPPORTED BROWSERS:

- **MSIE:** Microsoft Internet Explorer (Works only in Windows)
- **MSEDGE:** Microsoft Edge (Works only in Windows 10+)
- **FIREFOX:** Mozilla Firefox (Must be installed)
- **CHROME:** Google Chrome (Must be installed)
- **OPERA:** Opera Browser (Must be installed)
- **SAFARI:** Aple Safari (Default browser in Mac OS, must be installed in other OSes)
- **PHANTOMJS:** A headless WebKit browser.
- **HTMLUNIT:** HTMLUnit is a GUI-less browser made in pure Java. Supports Javascript.
- **HTMLUNIT_FF27:** A HTMLUnit browser with Mozilla Firefox 27 functionality emulation.
- **HTMLUNIT_IE11:** A HTMLUnit browser with Internet Explroer 11 functionality emulation. Supports ActiveX in Windows.

MICROSOFT EDGE WEBDRIVER:

By default, Open Test Suite is not shipping with the Microsoft Edge WebDriver. In order to support the Edge browser, you have to download the WebDriver, you can download it from here: https://www.microsoft.com/en-us/download/details.aspx?id=48212

"Open Test Suite" FAILED TESTS AND ERROR INFORMATION:

When a *Test* fails, the failed *Action* will be displayed next to the Test. The error information starts with two numbers inside of brackets "[]" separated by a ":".
The first number represents the Test number where the Action failed and the second one the failed Action number ([test_number:action_number]).
By double clicking on the error in the Test Player, loads the Test in the Test Editor and highlights the failed Action automatically.

All other errors and debug information is shown in the "**Console**" tab..

BEST PRACTICES:

As you know, Open Test Suite is based on Selenium, which is a powerful browser automation tool. It can handle almost every situation, as long as the Tests are of good quality. The Test quality depends on how well you handle the different situations on a

website. For simple *static pages* this can be done with ease, but in more *complex and dynamic* pages you have to let Selenium know what it has to do in the different situations, when to act and when to wait. As in dynamic pages the content is changed in real-time, the exact time moment when something changes or happen can vary, so clicking around the page without waiting for a specific state of an element or text is not an option or will be a matter of luck for it to happen (We don't want that).

Luck is not an option, think for the worst case scenario: On dynamic websites, you can write a Test and the Test may finish successfully, but as long as you don't wait for the correct elements, whenever in another moment something takes a bit more time, the whole Test can be marked as Failed, when everything is just fine.

Wait for something, not just wait around: The use of static waiting (*wait for x seconds*) is heavily discouraged as some times something might take longer than other times, instead wait for an element, text, etc to be present.

Wait more than necessary than not enough: Dynamic waiting has no mayor time impact on the Tests as when the circumstances meet, it just continues right away. So even if you do a dynamic waits more than necessary it has barely any impacts on the Tests, and minimizes the chances for a Test to fail when it shouldn't.
Finally it is recommended to have a look at the "Test Examples and Demos" which you can find in the next point.

TEST EXAMPLES AND DEMOS:

You can access a list of Test Examples and Demos visiting this Google Drive shared folder: https://goo.gl/t26ief

TIPPS & TRICKS:

HTTP Authentication from URL: Sometimes you need to Authenticate on an HTTP Auth Form and Selenium cannot handle that. Use this pattern instead:
http://username:password@www.url.com

INSTALL WEB SERVER AS SERVICE:

You can install Open Test Suite Server as Service in Windows using the provided batch files from the installation directory.
There are in total 4 batch files ("webserver_service_install","webserver_service_uninstall","webserver_service_start" and "webserver_service_stop"). Each of them are used to install/uninstall and start/stop the service. You have to run them using Administrator privileges. It may be possible that you will have to navigate with the CMD to the installation directory and execute the batch from there.

RESOURCES AND LICENSES:

Most of the image resources have been downloaded from: http://www.fatcow.com
And are licensed under a Creative Commons Attribution 3.0 License.
Some of the resources have been downloaded from: http://www.medialoot.com
All images and resources are Copyrighted by their respective owners.
This software is intended for personal use. For redistributing this Software please look for the licenses of the included 3rd party libraries and binaries inside of the "*/lib*" and "*/servers*" folders. License compatibility to do so is not guaranteed.
(**Please note:** Some licenses and readme's are missing in this package, though they are still applying. Those licenses have to be found manually using other sources.)

INFORMATION ABOUT "Open Test Suite":

Open Test Suite 1.0
Programmed by **Robert Kay Koszewski**.
For support or feature requests, drop an email to "**rkkoszewski@gmail.com**".

9.2. Web Server Interface Manual

QUICK USER MANUAL FOR "Open Test Suite Web Interface"

"Open Test Suite Web Interface" is aiming to be a passive testing platform, which allows to run Test Cases in a determined interval and notifying the user in case of mayor issues.

QUICK START GLOSSARY

In order to understand all of the concepts that "Open Test Suite Web Interface" we need to clarify some frequently used terms:

- **Quick Test:** Is a Test type where the user uploads a temporary Test to the server to run it. This Test will be never saved on the server and is meant for one-time testings.
- **Global Test:** A Global Test is loaded and launched automatically by the server. Global Tests are Tests saved on the server and can be managed in the "**Test Manager**" section.
- **Test Case:** A Test Case is basically one Test File which can have one ore more sub-Tests.
- **Test Health:** The Test Health is determined by the number of failed Tests in a Test Case.

MAIN PARTS:

"Open Test Suite Web Interface" has 6 main parts which can be accessed from the Main Menu. Those will be described next:

Main Menu:

- **Dashboard:** Here is the place where everything starts. There you can have a brief overview of the Test Statuses and have a general Server Status overview.
- **Test Player:** Here you can run "Quick Tests" on the server and see the current Test progression of the current running Test (It can be a running *Quick Test* or *Global Tests*). You can Start/Stop "Quick" and "Global Tests" from there too. In order to run a "Quick Test" you have to stop the "Global Tests", then you can upload a Test from your computer, optionally force a specific browser, and run the Test.
- **Test Reports:** Here you can see an overview of all Tests and their Health Status. From there you can access the Detailed Test Reports (Which will be described later in this manual) and the Server Health Status Reports.
- **Test Manager:** Here you can manage all the Tests of the Server. Use "Add new Test" to add a new Test Case to the "Test List". Once a Test Case is added then you can "Manage" it or load it into the Test Player by clicking "Run Now". From here

you can Start/Stop the Global Tests and view a "Realtime Test Queue" to know when the next Test Case will be Queued to the "Test Player".

- **Settings:** Here you can view and configure all settings for the Open Test Suite Server.
- **Help:** Here you can read information and help about Open Test Suite Desktop and Web Interface.

TEST & SERVER HEALTH:

There are 3 possible Health states: Healthy or Good, Average and Critical. These are used to simplify the process of identifying possible problems or elements to review.

Test Health:

- **Healty:** When a Test Case has 0 Failed Tests, then its Health is considered as Healthy.
- **Average:** If the number of Failed Tests is higher than 0 then it is considered as Average.
- **Critical:** When a Test Case has more or equal Failed Tests than the "Minium Number of Test Fails to be Critical" which can be configured in the "Settings" menu, then the Test Case is considered to be in Critical Health.

The Test Health status can be reset when desired, for example when a Test Case got identified and fixed, to be ready for the next time an issue arise.

Server Health:

OTS Web Interface is constantly monitoring itself, to notify the user on possible system instabilities or other issues. These are the possible Server Health Statuses:

- **Good:** When there were no issues happening on the server.
- **Average:** The Average Status will be shown when a warning happened on the Server, like CPU and RAM spiking during longer time periods, etc.
- **Critical:** This status will be shown when unexpected behaviour happens on the server, like Tests which cannot be loaded, or other IO issues which prevent the normal work of the Tests.

TEST REPORTS: TEST REPORT DETAILS

In the Test Reports section you can see a list of Tests and access to its Test Report Details where you can see more detailed information about the Test Case runs. It has several parts:

Overview Counter:

Here you can see an Overview counter. This counter will increase with each Test Run and marked as Clean Test Run (When the whole Test Case finished whichout any errors), or Tests With Failures (When one or more errors happened during a Test Session). This counters are just meant for indicative purposes and can be reset in any moment. They will increase while unattended testing, and when the User has checked the errors, then he/she can reset the counters so when an other error occurs he/she will be notified.
Please note: Resetting the counter wont reset any report data.

Quick Overview Charts

There are two overview charts. The "Test-Fail Report" chart and the "Performance Report" chart. Each of them are limited to the last 20 entries. More data can be generated in the "Detailed Report Generator".

Test Status Timeline:

In the Test Status Timeline you can see the last 50 Test Statuses. If a Test failed, you can see there detailed information of what failed, and when a camera is visible next to the error, by pressing on it, you can see a screenshot taken right when the error happened.

Detailed Report Generator:

Here you can generate a detailed report of the desired Time-Frame. You can have a general overview leaving the *day* or the *month* on "--". It will generate a PDF which can be just viewed or downloaded.

TEST MANAGER: MANAGE TEST

From the Test Manager you can upload or manage your Tests. You can enable/disable Tests directly from the Test List. A disabled Test won't be executed, but will remain on the server for future use and its report data won't get lost. Now we will go to the "Manage" section of a Test:

In the Test Manage section you can Enable/Disable the Test, change its name, force a different browser than defined in the test and set its Cron Expression. All those four elements will be saved when you press the "Save Changes" button. You can upload there your Test File. No data nor reports will be lost during this process. From here you can delete the whole Test Case, but bear in mind that all data and reports will be lost permanently.

As the Test Suite server is designed to be running on longer term, the "Test Report Database" may grow to huge sizes. In case it gets too big, you can reset it from here, and it will be regenerated from zero.

SETTINGS

In the Settings you can set the "Test Fails to be Critical", "Email Notifications", setup a Mailgun Account or SMTP, Enable/Disable/Change the User Authentification and Restart or Shutdown the Server (Just the Software, not the OS)

All theese settings are explained and should be auto-explanatory while interacting with them.

TEST EXAMPLES AND DEMOS:

You can access a list of Test Examples and Demos visiting this Google Drive shared folder: https://goo.gl/t26ief

INFORMATION ABOUT "Open Test Suite":

Open Test Suite 1.0
Programmed by **Robert Kay Koszewski**.
For support or feature requests, drop an email to "**rkkoszewski@gmail.com**".