Trabajo Fin de Grado
Ingeniería de Telecomunicación

Evaluación de herramientas para la detección automática de errores en aplicaciones software

Autor: José Manuel Moreno Poveda

Tutor: Francisco José Fernández Jiménez

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Grado
Ingeniería de Telecomunicación

Evaluación de herramientas para la detección automática de errores en aplicaciones software

Autor:
José Manuel Moreno Poveda

Tutor:
Francisco José Fernández Jiménez
Profesor Colaborador

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Proyecto Fin de Grado: Evaluación de herramientas para la detección automática de errores en aplicaciones software

Autor: José Manuel Moreno Poveda

Tutor: Francisco José Fernández Jiménez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019 El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Me gustaría dedicar unas líneas para agradecer a todos aquellos que de un modo u otro han estado a mi lado para apoyarme a lo largo de esta larga travesía.

Me cabe hacer una mención especial a mis padres, Isabel y Antonio Jesús, los cuales me apoyaron desde el principio y me dieron fuerzas para hacer este viaje mucho más ameno. Su ayuda incondicional y la educación recibida por su parte han hecho de mí lo que hoy en día soy.

No quiero olvidarme del resto de mi familia, los cuales me apoyaron continuamente para que este día llegara.

Por último mencionar a mi tutor de este proyecto, Dº Francisco José Fernández Jiménez, el cual confió en mí para llevar este trabajo acabo.

José Manuel Moreno Poveda

Sevilla, 2019

Resumen

Con el paso del tiempo, las aplicaciones, tanto webs como móviles, están evolucionando a pasos agigantados lo que conlleva una mayor complejidad de estas. Además se unen otros factores como recursos disponibles, modificaciones y nuevas funcionalidades a implementar, tiempo de entrega de dichos desarrollos,… Es una evolución constante sustentada en base a unas limitaciones para la que hay que estar preparados. El objetivo de este trabajo es dar a conocer los diferentes métodos para detectar posibles fallos y vulnerabilidades de forma automática de nuestras aplicaciones, para crear software de calidad que nos aporte mayor tranquilidad sobre las principales funcionalidades del producto final.

A la vez mediante el primer objetivo lo que se consigue es una reducción en todos los recursos disponibles, como puede ser el tiempo de entrega, recursos monetarios,…

Abstract

Over time, the applications, both web and mobile, are growing by leaps and bounds which entails a bigger complexity of these. In addition other factors such as available resources, modifications and new functionalities to be implemented, delivery time of these developments ... It is a constant evolution based on some limitations for which we must be prepared. The objective of this work is to present the different automation methods to detect possible failures and vulnerabilities of our applications to create quality software that gives us more security about the main functionalities of the final product.

At the same time through the first objective what is achieved is a reduction in all available resources, such as delivery time, monetary resources …

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

En este el primer capítulo se van a tener en cuenta una serie de apartados, en concreto cuatro bastante importantes para la contextualización de este trabajo de fin de grado.

El primer apartado será en referencia al porqué de la elección de este tema. Se va a exponer de forma breve el problema de no contar con un proceso de calidad durante el proceso de desarrollo de la propia aplicación. A partir del problema expuesto, veremos la motivación por la que se ha elegido este tema y haremos una breve explicación de que es QA (Aseguramiento de la Calidad / Quality Assurance).

En el segundo apartado se va a contextualizar la investigación realizada a la hora de la elaboración de este trabajo. En este punto se verán dos grandes temas diferenciados, las aplicaciones software y las pruebas. En cada uno de estos puntos se hará una breve explicación y posteriormente se verá alguna clasificación en tipos.

En el tercero se expondrán y se hablará concisamente sobre los diferentes objetivos que se pretenden alcanzar a la hora de su realización. Se intentará motivar al lector para que siga una cultura de calidad cuando trabaje sobre algún proyecto.

En el último apartado se va a hablar sobre la organización de este trabajo fin de grado, con el fin de adelantar al lector una breve explicación sobre cada uno de los apartados de este.

1.1 Propósito y Motivación

En la industria del desarrollo software cada vez es mayor la preocupación por la calidad de sus productos, ya que cada vez son más complejos este tipo de proyectos. En este proceso de calidad se tienen varios objetivos que tienden a fundirse en uno sólo, como es la entrega de productos estables y de calidad para así cumplir con los requisitos y expectativas del cliente. El gran problema que había desde hace un tiempo atrás eran las entregas grandes pero sin pasar por un proceso para verificar este producto, lo que conllevaba a productos más inestables, con más posibilidades de errores y fallos una vez subidos a producción. Es decir, se buscaba la cantidad y "rapidez", y se descuidaba el aspecto de encontrar errores y posibles fallos dentro del proceso de desarrollo. Con el paso del tiempo la industria del desarrollo se ha concienciado que este proceso es vital, y a la larga esto deriva en un ahorro tanto de tiempo como de presupuesto.

El principal propósito de la realización de este trabajo de fin de grado es conocer más a fondo los diferentes tipos de pruebas aconsejables a definir, desarrollar y ejecutar sobre nuestras diferentes aplicaciones de forma automática, las cuales dividiremos en dos grandes grupos para este fin, las aplicaciones web y las móviles.

Con la realización de las pruebas que se van a explicar durante este trabajo de fin de grado se favorecerá la detección de errores (bugs) en la aplicación durante el proceso de desarrollo de la misma. A la larga, aunque haya que tener un equipo especializado en calidad, esto resultará beneficioso en todos los sentidos ya que no es lo mismo encontrar un error durante el proceso de desarrollo que un error cuando la aplicación está publicada y accesible para todos los usuarios. Esto va a promover el ahorro de recursos y se construirá una aplicación estable para el usuario final, lo que supondrá más beneficios para el cliente. *"No es raro que el coste de las pruebas del software supongan un 40% del coste total de desarrollo del proyecto"* [1].

En gran parte nos vamos a centrar en la automatización de estos tipos de pruebas. Además de esto, nos adentraremos en el mundo de las herramientas y frameworks utilizados para la automatización de este tipo de pruebas y hablaremos sobre algunas de ellas diciendo sus principales ventajas para guiar al lector en su elección y que escoja la más afín a sus necesidades. Se propondrán casos prácticos de los diferentes tipos de pruebas, los cuales podrán usarse como guía para el lector.

Este proceso de automatización será beneficioso a la hora de comprobar la funcionalidad existente en la aplicación y que con nuevos desarrollos siga funcionando correctamente, es lo que se conoce como regresión. Mediante estas técnicas se mantendrán controladas especialmente las funcionalidades principales de la aplicación. Además, de esta forma se evitará la realización de tareas repetitivas.

Es necesario balancear el testeo manual y el automático. Primeramente, hay que hacer especial énfasis en el testeo manual de la aplicación y de los diferentes nuevos desarrollos. Seguidamente, habrá que ir implementando nuevas funcionalidades a los test automáticos, para que se comprueben de manera periódica nuevas funcionalidades y las ya existentes.

Ilustración 1 Proceso de calidad (QA)

https://www.cybage.com/product-engineering/testing-and-qa/qa-processes

En la anterior imagen podemos ver a grandes rasgos todo lo que incumbe este proceso de calidad, desde los diferentes modelos a implementar que ocupan el interior de la circunferencia, pasando por el proceso en sí del equipo de testing y por último las diferentes actividades que se dan durante estas fases del proyecto.

La principal motivación que me ha llevado a escoger este tema para la realización de este trabajo es porque llevo alrededor de dos años inmerso laboralmente en este sector comúnmente conocido como QA (Aseguramiento de la Calidad / Quality Assurance).

Desde mis inicios hasta el día de hoy he pasado por varias empresas en las que he cumplido con diferentes roles a medida que me he ido desarrollando en este campo.

Al principio sólo diseñando y ejecutando pruebas sencillas de forma manual contra aplicaciones web para detectar posibles fallos y reportarlos a un equipo de desarrollo el cual trabajaba conjuntamente con el equipo de testing.

Posteriormente introduciéndome totalmente en el mundo de las pruebas automáticas, más concretamente en pruebas automáticas funcionales contra aplicaciones móviles, de las cuales hablaremos en uno de los últimos puntos de este trabajo. Un trabajo más técnico en el que es necesario desarrollar habilidades de programación en diferentes lenguajes posibles y la utilización de diferentes frameworks y herramientas para su desarrollo.

Finalmente en mi actual etapa me encuentro inmerso en el mundo de las pruebas tanto manuales como automáticas contra aplicaciones web. Además de las pruebas funcionales también me estoy desarrollando en otro tipo de pruebas que van desde pruebas de seguridad para evitar posibles vulnerabilidades en nuestras aplicaciones, hasta pruebas de rendimiento para la creación de un software de calidad y ágil.

Veremos además que no sólo de este tipo de pruebas vive el campo de la calidad software y es por eso por lo que me ha sido de una motivación extra, para investigar y aprender sobre nuevas técnicas y pruebas de las cuales no tengo experiencia en lo laboral.

Es por todo esto por lo que he escogido este tema, que es actualmente y desde hace un par de años mí día a día.

Antes de entrar en más detalle acerca de los tipos de pruebas, herramientas y automatización, daremos una breve explicación sobre qué son las pruebas software y los motivos por los cuales se aconseja su uso, cada vez más creciente. Esto se avanzará en el siguiente punto de este capítulo, el cual sería básicamente para poner en contexto dicho trabajo en el que también se hablará de otros temas como qué son la aplicaciones software, haciendo la división entre web y móvil como antes se mencionó, y el por qué hoy en día son tan importantes en nuestras vidas.

1.2 Contexto de la investigación

El marco del contexto en el que se va a desarrollar este trabajo será el de las aplicaciones software, tanto aplicaciones web como móviles, haciendo especial hincapié en las primeras mencionadas ya que la evolución en el testing automático contra este tipo de aplicaciones ha sido mayor a día de hoy, lo que conlleva a más información disponible, más herramientas, más técnicas…

Con el fin de ponernos en situación antes de empezar a hablar de pruebas, herramientas, automatización y todo lo que conlleva el sector del testing como tal, comenzaremos a hablar sobre las aplicaciones en las que vamos a desarrollar este trabajo de testeo. Vamos a ver, a grandes rasgos, que son estas aplicaciones, algunas diferencias entre ellas y una de las clasificaciones posibles que se pueden hacer.

1.2.1 Aplicaciones software

Para ponernos en situación, vamos a definir primeramente que es una aplicación web y una aplicación móvil, viendo así las principales diferencias entre ellas.

Echando un vistazo por el portal Wikipedia, podemos observar la siguiente definición de aplicación web: "En la ingeniería de software se denomina aplicación web a aquellas herramientas que los usuarios pueden utilizar accediendo a un servidor web a través de internet o de una intranet mediante un navegador. En otras palabras, es un programa que se codifica en un lenguaje interpretable por los navegadores web en la que se confía la ejecución al navegador" [2].

Viendo la definición de aplicación móvil en el mismo portal nos podemos dar cuenta de algunas características que las hacen diferentes: "Una aplicación móvil o app, es una aplicación informática diseñada para ser ejecutada en teléfonos inteligentes, tabletas y otros dispositivos móviles. Las aplicaciones permiten al usuario efectuar un conjunto de tareas de cualquier tipo —profesional, de ocio, educativas, de acceso a servicios, etc. —, facilitando las gestiones o actividades a desarrollar" [3].

Por un lado tenemos las aplicaciones web, que son aplicaciones software desarrolladas en múltiples lenguajes (como puede ser HTML, XHTML, PHP, CSS…), de las cuales podremos disfrutar de sus contenidos simplemente accediendo desde cualquier navegador sin importar el sistema operativo del que dispongamos. En el otro, tenemos las aplicaciones móviles, que también son aplicaciones desarrolladas en múltiples lenguajes como las anteriores pero que por lo general se encuentran disponibles para el usuario a través de distintas plataformas dedicadas a la distribución de estas, las cuales suelen ser propias de los sistemas operativos del dispositivo móvil en cuestión.

Otra característica a remarcar dentro de las aplicaciones móviles es que tienen diferentes características a las web para poder funcionar en este tipo de dispositivos, ya que estos suelen tener más limitaciones, lo que conlleva una menor capacidad de procesamiento y a su vez menos capacidad para almacenar datos [4].

También es diferente la forma en la que dichas aplicaciones actualizan su contenido. Cuando una aplicación móvil tiene alguna actualización, para poder usar esta nueva versión nos la tendremos que descargar desde la plataforma de distribución de nuestro dispositivo. Por el contrario, cuando una aplicación web se actualiza los usuarios no nos tenemos que descargar nada ni seremos consciente en gran medida de esta actualización. La mayoría de las aplicaciones web no requieren instalación por parte del usuario.

Se pueden hablar de otras muchas diferencias, como por ejemplo el propósito de unas y de otras, la forma de crearlas, el pago por disponer de los servicios de esa aplicación… Con las que hemos nombrado aquí arriba nos podemos dar cuenta de que las diferencias entre unas y otras son claras y los procesos y herramientas dedicadas, como por ejemplo en todo lo referente al testing automático y sus herramientas en este caso, deben de ser diferentes.

Ahora vamos a ver una de las posibles clasificaciones dentro de cada una de ellas, empezando por las aplicaciones web, para ver los diferentes tipos que nos podemos encontrar.

1.2.1.1 Tipos de aplicaciones web

Dentro de las aplicaciones web, vamos a hacer una clasificación en 2 grandes grupos diferenciados, aunque luego nos detendremos en el segundo grupo y haremos una subdivisión de este.

1. **Aplicación web estática**

   La principal característica de estas aplicaciones es que muestran poca información al usuario y no suelen realizarse muchos cambios.

   Este tipo de aplicaciones suelen combinar los lenguajes de programación HTML y CSS aunque a veces es usual encontrarse con diferentes objetos con movimiento como pueden ser GIF animados, videos, banners…

   Una de las principales ventajas de este tipo de aplicación es que pueden resultar muy económicas en comparación con las siguientes. Además, también son más rápidas de desarrollar, ya que carecen de funcionalidades.

   Algún ejemplo sencillo de este tipo de aplicaciones puede ser una web de presentación de una empresa con sus correspondientes datos de contacto.

2. **Aplicación web dinámica**

   Son aplicaciones bastante más complejas que las anteriores. Utilizan bases de datos para actualizar los contenidos de esta misma a medida que el usuario va introduciéndose en la aplicación. Permite al usuario una mayor interacción con la aplicación. Las posibilidades, en cuanto a diseño y desarrollo de este tipo de aplicaciones, son infinitas.

   En cuanto a lenguajes de programación a día de hoy existen muchísimos que se pueden nombrar, como es el caso de PHP o ESCMAScript.

   Un ejemplo de este tipo de aplicaciones puede ser cualquier foro que nos encontremos en la web.

   A partir de este gran grupo, podemos sacar múltiples clasificaciones. Aquí se expone una de ellas [5]:

   a. **Tienda virtual o comercio electrónico**

   Son comúnmente llamadas e-commerce. Su desarrollo se complica ya que se incluye la tarea del pago electrónico a través de múltiples posibilidades, como tarjetas de crédito, plataformas

como PayPal, tarjetas propias de la compañía dueña de la aplicación…

Todas las aplicaciones en referencia a tiendas online son un ejemplo de estas, podemos nombrar todas las del grupo Inditex.

b. Portal web app

En el portal Yeeply la definen de la siguiente manera: "Con portal nos referimos a un tipo de aplicación en el que la página principal permite el acceso a diversos apartados, categorías o secciones. Puede haber de todo: foros, chats, correo electrónico, un buscador, zona de acceso con registro, contenido más reciente, etc."

c. Aplicación web animada

Las aplicaciones animadas nos permiten presentar diversos contenidos los cuales incorporan efectos animados, asociado a la tecnología FLASH. Esto complica su desarrollo y pero potencia la creatividad.

Hay un gran inconveniente a nombrar en este tipo de aplicaciones y es el relacionado con el posicionamiento web y su optimización. Es un punto débil a tener en cuenta ya que los buscadores no pueden interpretar de forma correcta la información.

d. Aplicación web con gestor de contenidos

Este tipo de aplicaciones necesitan un gestor ya que se necesitan actualizar constantemente. Estos gestores suelen ser fáciles de utilizar.

Algunos ejemplos de estos gestores de contenidos son WordPress, que es el gestor más extendido y es software libre, también se puede tener en cuenta Joomla y otro que cada vez está cogiendo mayor fuerza, Drupal, recomendado por su adaptabilidad.

Un ejemplo claro de este tipo de aplicaciones pueden ser las páginas de noticias y medios de comunicación.

1.2.1.2 Tipos de aplicaciones móviles

Ahora, dentro de las aplicaciones móviles, vamos a ver una de las posibles clasificaciones. Esta clasificación está compuesta de 3 grandes grupos diferenciados que son los siguientes:

1. Aplicaciones nativas

Este tipo de aplicaciones están creadas bajo un lenguaje y entorno de desarrollo específico, esto aporta una gran fluidez en su funcionamiento, además de estabilidad en su funcionamiento a partir del sistemas operativo para la que fue creada.

La gran mayoría de estas aplicaciones no necesitan estar conectadas a la red para su funcionamiento lo cual también supone un punto a favor de estas aplicaciones.

Además, se permite la publicación de estas en tiendas para su posterior distribución.

No todo son ventajas, como nombramos al principio funcionan de manera fluida para el sistema operativo para el que fueron creadas pero, y ¿qué pasa con los demás dispositivos con diferentes sistemas operativos? La respuesta a esta cuestión es que no se pueden utilizar.

Además, otras cosas a tener en cuenta son el costo para distribuirla en una tienda y la necesidad de ser aprobada para su publicación.

2. **Aplicaciones web**

   Estas están creadas con lenguajes dedicados al desarrollo web como pueden ser HTML, CSS, ECMAScript… El gran beneficio de estas aplicaciones es la posibilidad de ser usadas desde cualquier dispositivo sin importar que sistema operativo tengan, ya que con el navegador nos es suficiente.

   Otras ventajas en referencia a las anteriores es que su costo es mínimo y que no necesita la aprobación de nadie para su publicación.

   Las mayores desventajas que se pueden nombrar son que no utilizan ninguna plataforma para su distribución, en contraposición de las nativas, y que no usa de una manera óptima los recursos del sistema, lo cual es un aspecto a tener en cuenta.

3. **Aplicaciones híbridas**

   Estas aplicaciones, como su propio nombre indica, tienen características de ambas aplicaciones anteriormente nombradas. Se utilizan lenguajes de desarrollo web pero cambiando el framework, el cual es dedicado para la creación de este tipo de apps.

   Son aplicaciones multiplataforma y en estas sí que se permite su distribución a través de las tiendas.

   Además el costo de estas suele ser inferior al de las nativas.

   La mayor desventaja es que son un poco desconocidas y se carece de información en muchos casos para su elaboración.

Una vez hemos visto que es una aplicación software y todo lo referente a sus dos grandes grupos, las aplicaciones web y las aplicaciones móviles, vamos a pasar a ver que son las pruebas software. Además de ver la definición de estas como tal, también nos detendremos y se expondrá una clasificación a grandes rasgos de las pruebas más importantes durante el ciclo de desarrollo software, en base al nivel de testeo como criterio escogido.

En el portal Wikipedia se definen de la siguiente forma: “Las pruebas de software (en inglés software testing) son las investigaciones empíricas y técnicas cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto a la parte interesada o stakeholder. Es una actividad más en el proceso de control de calidad” [6].

Como vemos en la definición anterior, es una actividad más dentro del proceso de calidad. Se puede decir que es de las más importantes, si no la que más. No es más que la ejecución de un sistema o de algún módulo en concreto de este en unas circunstancias conocidas, en las cuales debemos conocer tanto la entrada como la salida esperada con el fin de evaluar el resultado dado.

Estos test deben ser atómicos. Deben ser independientes unos de otros y se debe conocer en cada momento la salida esperada sin dejarlo en manos de alguna lógica provocada por alguna entrada, conocida o no, en el sistema.

Además de esto nos hemos centrado en que debemos conocer la entrada y la salida previamente a la ejecución de estas pruebas. Siempre que se ejecute un test debe dar el mismo resultado independientemente de la situación y de las condiciones de su ejecución.

Con estas características mencionadas, el mejor ambiente para realizar este tipo de pruebas a nuestra aplicación es un ambiente totalmente independiente del desarrollo de la aplicación. De esta forma se consigue la no contaminación por parte de los desarrolladores y por consiguiente la objetividad acerca de los resultados de dichas pruebas.

Con estas pruebas se pretende evaluar la calidad de nuestra aplicación software, con el objetivo de mejorarla y desarrollar una aplicación estable en la medida de lo posible.

1.2.2 Tipos de pruebas

Una vez que hemos hecho una pequeña introducción acerca de qué son las pruebas software, vamos a hacer una clasificación de estas por nivel de testeo:

Ilustración 2 Pirámide testing

1. **Pruebas unitarias**

   Son las primeras pruebas a tener en cuenta, las de más bajo nivel.

   Estas suelen llevarlas a cabo los propios programadores una vez están implementando una solución a algún error encontrado o desarrollando alguna nueva funcionalidad.

   Con ellas se prueban 'trozos de código' bastante limitados. Suelen ir contra métodos y se suelen hacer de forma aislada para comprobar el correcto funcionamiento de estos.

En un apartado más adelante se desarrollarán más este tipo de pruebas. Además de ello veremos herramientas útiles para la automatización de estas pruebas.

2. **Pruebas de integración**

   Este tipo de pruebas comprueba la interacción de diferentes unidades de software. Hace referencia al funcionamiento en conjunto de varios módulos no de forma aislada como en el anterior caso.

   Este tipo de pruebas además de tener la fuente de error del fallo en el desarrollo, también tiene la posibilidad de deberse a fallo en el entorno en el que está desplegado.

3. **Pruebas de sistema**

   El objetivo de este tipo de pruebas es verificar el funcionamiento del software al completo, es decir, de todos los diferentes módulos funcionando a la vez e interactuando unos con otros. Es por ello por lo que es conveniente realizar este tipo de pruebas después de las de integración.

   Aquí es donde se comprueba la seguridad, velocidad, fiabilidad y otras muchas características del software.

4. **Pruebas de aceptación**

   Este tipo de pruebas se realizan para comprobar si el software desarrollado cumple con las expectativas del cliente que lo mandó desarrollar.

   Estas pruebas se suelen ejecutar en último lugar para verificar completamente que el producto es apto.

1.3 Objetivos

En este tercer punto, se van a exponer los diferentes motivos que me han llevado a escoger dicho tema para desarrollarlo como mi trabajo de fin de grado.

Como se ha ido comentando en las anteriores secciones, las aplicaciones software han cobrado mucha importancia en nuestras vidas. Día a día, aún sin ser consciente de ello, estamos en constante manejo de ellas ya sean web, móviles… todo ello en diferentes dispositivos, de los cuales en la actualidad disponemos de un amplio abanico.

Ya que estamos en constante uso de ellas, es lógico que estas nos proporcionen unos servicios con una cierta calidad. Para cumplir este objetivo, habría que pasar por un proceso de calidad de forma continuada en el tiempo. Para cubrir este proceso de calidad tendremos que pasar por una serie de fases y aplicar ciertos tipos de pruebas en nuestros proyectos. De aquí sale el primer objetivo de este trabajo:

1. **Exponer y ver la importancia de cada uno de los tipos de pruebas que se van a exponer en este trabajo de fin de grado.**

   Se verán las pruebas software más usadas en las empresas a lo largo del proceso de calidad, aunque hay otras muchas más. Se definirán y se verá la importancia de cada una con el objetivo de inculcar la realización de estas sobre nuestras aplicaciones.

La detección de errores en las aplicaciones software puede ser una tarea laboriosa y a veces se puede volver repetitiva ya que constantemente estas aplicaciones están evolucionando e implementando nuevas funcionalidades, lo cual implica no sólo comprobar las nuevas funcionalidades si no también probar que los módulos anteriores siguen funcionando como hasta entonces y la integración de estos nuevos módulos con la funcionalidad ya existente. Es de aquí de donde sale el segundo objetivo a perseguir con este trabajo:

2. **Estudiar las diferentes técnicas para automatizar las diferentes clases de pruebas en sitios web y móviles**.

   A partir de ello, se verán posibles herramientas para llevar a cabo este trabajo con distintas características y servicios a ofrecer, desarrollando las más implementadas. También se propondrá un ejemplo práctico para cada una de ellas con el fin de analizar resultados y de dar unas pequeñas pautas al lector sobre cómo realizar automáticamente este tipo de pruebas software.

Todo esto promueve un objetivo común y más ambicioso, el cual hoy día muchas de las empresas del sector de desarrollo software están implementando. Ya que las aplicaciones están en constante evolución y cada día son más complejas, es necesario pasar por un proceso de calidad para asegurarnos un proyecto competitivo en el mercado actual. Es por esto por lo que se nombra este objetivo:

3. **Concienciar al desarrollo de aplicaciones fiables y anticiparnos a los posibles fallos y vulnerabilidades haciendo un trabajo de calidad.**

   Además de muchas otras cosas, el que nuestra aplicación haga frente a diversos tipos de

pruebas con sus correspondientes objetivos, ayuda en gran medida a desarrollar aplicaciones competentes para el mercado. Al fin y al cabo, las empresas buscan beneficios a la hora de construir un proyecto, como es el de una aplicación software, y este es un buen paso para luchar con la competencia dentro de su mismo sector, ofreciendo calidad en sus productos.

El último objetivo es más a nivel personal. Ya que estoy desarrollando mi carrera profesional en este sector del testing, me gustaría promover la cultura de calidad y considero que es una buena forma de empezar esto haciéndolo aquí en este trabajo de fin de grado:

4. **Dar a conocer el mundo del testing y de la calidad software.**

   Ya que es a lo que me dedico desde hace un tiempo y es un campo que está comenzando a avanzar a pasos agigantados en estos últimos años. Muchas empresas lo están implementando y es un sector con muchas salidas y mucha demanda de profesionales en el mercado laboral actual.

Estos son los cuatro objetivos fundamentales en los que se basa mi trabajo de fin de grado, haciendo especial hincapié en los dos primeros mencionados y en el último a modo personal, ya que el tercer objetivo es bastante complejo y necesita de otros muchos factores y de evolución con el tiempo para que diese sus frutos.

1.4 Conclusiones

En la actualidad, casi todo el mundo posee un ordenador y/o un dispositivo móvil. La funcionalidad de estos se sustenta en su mayoría por aplicaciones software, tanto web como móviles. De ahí la importancia que han adquirido estas aplicaciones software y la competencia y el gran mercado que actualmente poseemos los usuarios de estas.

Estamos en continuo uso de ellas ya sea con fines laborales, diversión u otros propósitos y siempre buscamos algo novedoso o que cumpla mejor nuestras expectativas. Nos permiten programar nuestro día a día, realizar gran parte de nuestro trabajo, comunicarnos con personas en cualquier parte del mundo gratis…

Hemos podido comprobar la diversidad de ellas y mediante los ejemplos expuestos hemos podido ver que prácticamente todos hemos usado en alguna ocasión alguna de cada tipo aunque no seamos conscientes de lo que hay detrás.

Y si estas aplicaciones son tan importantes en nuestras vidas es en gran parte a la calidad de estas y a los servicios que nos ofrecen. Es aquí donde entran en juego las pruebas software y la importancia que cada vez más están tomando en el proceso de creación de una aplicación.

Es una parte muy importante del desarrollo software pero a la vez es una de las etapas más costosas en el proceso. No es de extrañar que el costo de este proceso pueda llegar a representar entre el 30% y el 50% del costo total del desarrollo del software [Myers, 2004]. Sin embargo un fallo en producción puede llegar a ser catastrófico y mediante este proceso se pueden evitar este tipo de imprevistos.

Las aplicaciones están desarrolladas para facilitarnos la vida y hacer más sencillas diversas tareas, entretenernos y otro sin fin de buenos motivos pero cuando fallan pueden traer consecuencias bastante graves las cuales debemos tener en cuenta.

Tenemos múltiples ejemplos de estas catástrofes y de lo que supone un mal control de la calidad del software, como por ejemplo:

- La desintegración de una sonda climática enviada a Marte por la NASA en el año 1999 en la cual hubo errores en el software correspondiente a la navegación que causaron la colisión de esta en la atmósfera de Marte. En este proyecto se invirtieron alrededor de 125 millones de dólares y por un error en dicho sistema que se pudo evitar mediante un proceso de calidad y de testeo, fracasó estrepitosamente.
- En 2012, otro error casi provoca la quiebra de la empresa Knight Capital. Esta empresa perdió en apenas media hora una cifra cercana a los 500 millones de dólares ya que por un fallo software los ordenadores empezaron a comprar y vender millones de acciones sin lógica ninguna.

Esto son sólo dos ejemplos de errores catastróficos pero podríamos nombrar muchísimos más con consecuencias bastante más graves como la exposición y el robo de datos personales y la pérdida de vidas humanas.

Es por esto y por otros muchos motivos por lo que nos podemos dar cuenta de la importancia de hacer un software de calidad y el por qué hay que dedicarle recursos y esfuerzo en esta laboriosa etapa.

Por estos motivos expuestos, es importante también la parte de automatizar las pruebas software de

diversas funcionalidades para así tener un control continuo sobre estos módulos y que las personas encargadas de realizar las pruebas manuales comprueben otro tipo de funcionalidades con el fin de encontrar todas las vulnerabilidades y fallos posibles en el sistema.

2 PRUEBAS UNITARIAS

En este segundo capítulo vamos a comenzar a ver las distintas pruebas a pasar en las aplicaciones software y como hacer este trabajo de forma automática. Más concretamente este capítulo tratará sobre las pruebas unitarias las cuales han sido mencionadas por encima en capítulos anteriores.

Vamos a definir este tipo de pruebas más a fondo, vamos a analizar sus características así como sus ventajas.

También hablaremos sobre posibles herramientas para realizar este trabajo de forma automática y explicaremos brevemente en qué consisten cada una de ellas y el porqué de elegir una u otra dependiendo de las necesidades de cada uno.

Al final se expondrá un ejemplo práctico sencillo sobre este tipo de pruebas, ya que este tipo de pruebas están más enfocadas a los desarrolladores para probar el código que ellos mismos implementan, no a los testers.

2.1 ¿Qué son las pruebas unitarias?

Son tareas de las que esencialmente se encargarán los propios desarrolladores. Les servirá para comprobar el correcto funcionamiento de la parte de código introducido así como ayudar a la refactorización de dicho código. Estas pruebas se suelen hacer sobre los diferentes métodos que implementa el desarrollador y por lo tanto no hay una persona más cualificada para hacer el trabajo que él mismo, ya que conoce que deben hacer los métodos que él mismo está implementando.

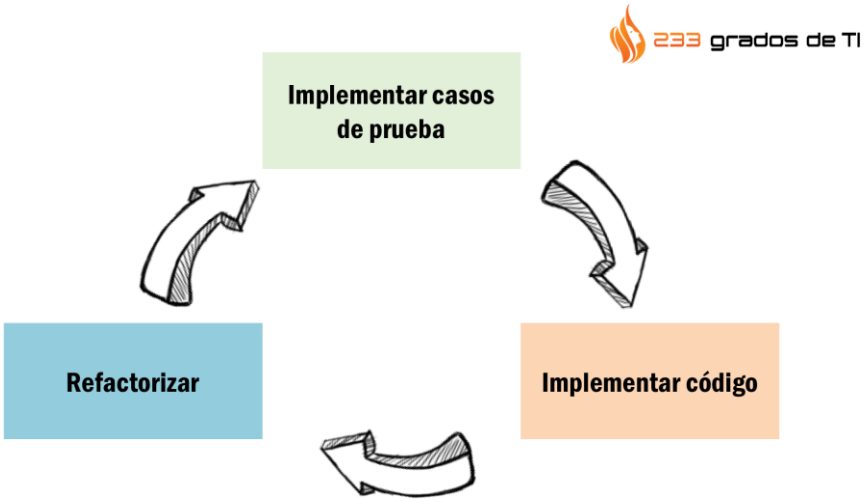


Ilustración 3 Desarrollo Dirigido por Test (Test Driver Development o TDD)

http://233gradosdeti.com/

Son pruebas a unidades de código totalmente independientes, aquí no se comprueba la funcionalidad conjunta de unos métodos con otros, se centra en examinar la correcta funcionalidad de un método en concreto. La superación de estos test de forma independiente por dos métodos no asegura la correcta funcionalidad entre ellos dos, esto se comprobaría con los test de integración que antes nombramos. Es por esto por lo que se tiene que reducir el número de precondiciones para poder llevar a cabo estos test, ya que tienen que ser lo más aislados posibles y que probar funcionalidades independientes.

El proceso desde que se definen los requisitos necesarios que se esperan del método probado hasta que se puede ejecutar el test unitario se podría dividir en tres partes, que es lo que se conoce como el patrón AAA (Arrange, Act y Assert), que son las siguientes:

1. **Arrange (Planear):** Se definen los requisitos que debe cumplir el código que se quiere testear.

2. **Act:** Es el propio proceso de creación de dichas pruebas unitarias teniendo en cuenta lo planeado en el paso anterior.

3. **Assert (Comprobar):** Es la etapa en la que se comparan los resultados expuestos y los previstos y se da un veredicto dependiendo de estos. Se puede validar el nuevo código implementado en caso de éxito de la prueba o se puede tener que refactorizar hasta que el error desaparezca. Estas pruebas facilitan esta tarea de refactorización haciendo el proceso mucho más liviano y reduciendo el número de iteraciones en el que se refactorizará el código.

Entre el paso 2 y 3 mencionados, irá el desarrollo del método a probar. A las personas que son nuevas en este tema les resultará raro que el primer paso para pasar estos test unitarios sea el de crear los test antes de crear el propio código, se pueden preguntar ¿Cómo probar algo que aún no se ha implementado? Pero para esto hay una explicación. El proceso es así porque si desde un primer momento definimos los requisitos que debe cumplir el nuevo código a implementar mediante estas pruebas, la tarea de desarrollo va a ser mucho más sencilla ya que sabremos todo lo que tiene que cubrir este y desde el principio podremos conocer todas las posibles casuísticas para evitar refactorizar el método en repetidas ocasiones.

Este proceso tiene un nombre, Desarrollo Dirigido por Test (Test Driver Development o TDD), el cual es un proceso de desarrollo software que se basa en la idea antes mencionada de desarrollar pruebas, codificar y refactorizar.

2.2 Características y ventajas

Las principales características que deben cumplir dicho tipo de pruebas son:

- **Automatizable**: Deben poderse automatizar para el no requerimiento de una intervención manual de forma que podemos hacer las pruebas de forma individual o grupal y será bastante útil para la integración continua.

- **Completas**: Son pequeños test sobre partes específicas de código pero su finalidad es cubrir dicho código al completo de forma individual.

- **Reutilizables**: No se deben crear pruebas que sólo se vayan a ejecutar una única vez dado que el código tiene que estar en constante comprobación para anticiparnos a los errores y encontrarlos lo antes posible. Además tiene que cumplir que en todas las ejecuciones el resultado de estas pruebas tienen que ser el mismo independientemente del orden en el que se ejecuten.

- **Independientes:** Esta independencia favorece la acotación del motivo de fallo, su localización y la forma en que se puede reparar.

Algunas de las principales ventajas de este tipo de test son:

- **Trabajo ágil, detección rápida de errores y reducción de costes/tiempo:** Te permite la detección temprana de errores de forma que se puede reescribir el código o corregir errores sin necesidad de rehacer todo el código. Facilita las tareas de depuración (debugging) ya que se basa en comprobar pequeños pedazos de código y estas pruebas se realizan periódicamente de forma repetitiva lo que ayuda en esta tarea de detección temprana de errores. Por estas razones se deduce que hay una reducción importante de tiempo, recursos y costo.

- **Calidad del código:** Al ser pruebas repetitivas y centrarse en funcionalidades concretas, se pueden detectar los errores con facilidad con lo cual se pueden reparar más fácilmente lo que conlleva a un código limpio y de calidad.

- **Favorecer la integración y los cambios:** Al estar divididos en bloques individuales es más simple la integración de nuevos desarrollos para hacer un código más completo o actualizar el que ya estaba, hace referencia a la frase de Julio César: “Divide y vencerás”.

- **Documentación:** Dichas pruebas nos pueden servir también como parte de la documentación ya que ahí se puede ver como se utiliza dicha pieza de código.

2.3 Herramientas utilizadas

En este tercer apartado del capítulo, vamos a hacer referencia a los frameworks para la automatización de estos test unitarios con una mayor relevancia. Vamos a ir nombrándolas y definiéndolas una a una y viendo sus diferentes características.

2.3.1 JUnit

Es un conjunto de bibliotecas que son utilizadas en el ámbito del desarrollo para hacer pruebas unitarias de aplicaciones Java. Fue creado por Erich Gamma y Kent Beck.

Ilustración 4 JUnit

https://junit.org/junit5/

Permite la ejecución de clases Java de manera controlada con el fin de evaluar los resultados que esta arroja ante una entrada dada y una salida previamente conocida. Esto sirve para comprobar el correcto funcionamiento de cada método. Devolverá acierto o fallo dependiendo de la salida que produzca el método y su posterior comparación con el resultado previsto.

Aparte de esto también es un posible método para el control de las pruebas de regresión, las cuales son necesarias cuando una parte del código ha sido modificada o se implementa nueva funcionalidad y se quiere ver si el nuevo código afecta, o no, negativamente a la funcionalidad total después de esta modificación.

2.3.2 TestNG

Es un framework de testeo para aplicaciones desarrolladas en Java. Fue creado por Cédric Beust. Es similar a la herramienta antes mencionada, JUnit, y aunque ambas son un framework para los test unitarios, esta última incorpora algunas funcionalidades que lo hacen más eficiente. No obstante está inspirado en JUnit y NUnit, del cual se hará una breve explicación más adelante.

Ilustración 5 TestNG

https://testng.org

Vamos a ver las principales diferencias entre estas dos herramientas de las que hasta ahora hemos hablado:

- TestNG admite la ejecución de pruebas en paralelo, mientras que JUnit no da esa opción a los usuarios.
- Los procesos de configuración de una prueba parametrizada en TestNG son bastante más sencillos.
- JUnit no es compatible con pruebas grupales, no se pueden administrar ciertas pruebas a un gran número de personas a la misma vez.
- TestNG soporta las pruebas de dependencia.
- Hay diferencias en las anotaciones para la realización de acciones antes, durante y después de la ejecución de los test.

2.3.3 NUnit

Definiendo brevemente lo que es NUnit podemos decir que es un framework dedicado al desarrollo y ejecución de pruebas unitarias para todos los lenguajes de programación .NET. Es una herramienta gratuita y de libre distribución. Fue creado por Charlie Poole y Rob Prouse, a la que luego se añadieron Jamie Cansdale, Gary Feldman, Charlie Poole y Michael C. Two.

Inicialmente fue desarrollado como JUnit, pero a día de hoy con las nuevas versiones se rehízo totalmente el framework aportando nuevas características y dando soporte multiplataforma. Tiene una interfaz similar a la de JUnit [7].

Ilustración 6 NUnit

https://nunit.org/

De esta herramienta cabe destacar que se pueden ejecutar test en paralelo, lo que no es posible con la herramienta a partir de la cual se creó esta, JUnit.

Para la realización y posterior ejecución de test unitarios con esta herramienta no es necesario escoger ningún tipo de proyecto en concreto.

Está desarrollado en C# y un lenguaje de programación orientado a objetos (OOP) que combina el poder de C++ con la simplicidad de Visual Basic.

2.3.4 Mocha

Es otro framework totalmente gratuito y de libre distribución (Open Source) que cada día está siendo más y más utilizado ya que es muy rico en características. Este puede ser implementado en NodeJS y cuenta con soporte de un navegador. Se usa sobre todo para hacer test unitarios a métodos implementados mediante el lenguaje de programación JavaScript.

Se ejecutan las pruebas en serie permitiendo sacar información mediante reportes flexibles y exactos. También es usual la implementación de pruebas asíncronas mediante esta herramienta.

Ilustración 7 Mocha

https://mochajs.org/

En muchos de los portales que dan información acerca de esta herramienta lo caracterizan o definen mediante tres palabras, “simple”, “flexible” y “divertido”.

Aquí se dejan algunas de las principales características de esta herramienta:

- Soporte para diferentes navegadores.
- Reportes de cobertura de código.
- Uso de librerías assert (para usar comparativas).
- Javascript API para ejecutar pruebas.
- Soporte de Debugger para NodeJS.

2.3.5 Yasca

Yasca es una herramienta creada por Michael V. Scovetta en 2007, especializada en analizar código que nos permite encontrar vulnerabilidades de seguridad, calidad en el código y rendimiento. Aprovecha las funcionalidades de otros plugins como pueden ser FindBugs, PMD, Jlint, PHPLint, RATS, CppCheck…

Ilustración 8 Yasca

https://en.wikipedia.org/wiki/Yasca

Soporta los siguientes lenguajes de programación:

- .NET (VB.NET, C#, ASP.NET)
- ASP
- C/C++
- COBOL
- ColdFusion
- CSS
- HTML
- Java
- JavaScript
- Perl
- PHP
- Python

2.4 Ejemplo de prueba unitaria

Para ver un ejemplo práctico de automatización de este tipo de pruebas unitarias vamos a utilizar la primera herramienta de la que antes hemos hablado, JUnit. Es la herramienta más popular por los desarrolladores dentro del sector de las pruebas unitarias.

Lo vamos a hacer sobre un sencillo método que vamos a implementar el cual se llamará suma, del cual por su nombre se puede deducir que va a sumar dos números enteros los cuales recibe como parámetros. Como se dijo en la introducción, no vamos a entrar mucho en pruebas complejas de este tipo ya que es un trabajo más propio de los desarrolladores, es por eso que se ha elegido este método sencillo que realiza la suma de dos números.

El lenguaje de programación escogido será Java, ya que es uno de los lenguajes más extendidos, de ahí que escojamos como herramienta JUnit.

Lo primero que tendremos que hacer será crear un nuevo proyecto. Este lo vamos a crear como proyecto Maven.

Para los que no sepan que es Maven, decir que es una herramienta para la gestión y construcción de proyectos Java. Se basa en XML. Utiliza un Project Object Modal (POM), el cual es un fichero escrito en lenguaje XML y es la principal unidad de un proyecto Maven. Contiene información acerca del propio proyecto desarrollado, fuentes, test, dependencias, plugins, versiones…

Este proyecto se crea en Eclipse escogiendo la pestaña File -> New -> Project y escogemos Maven Project:

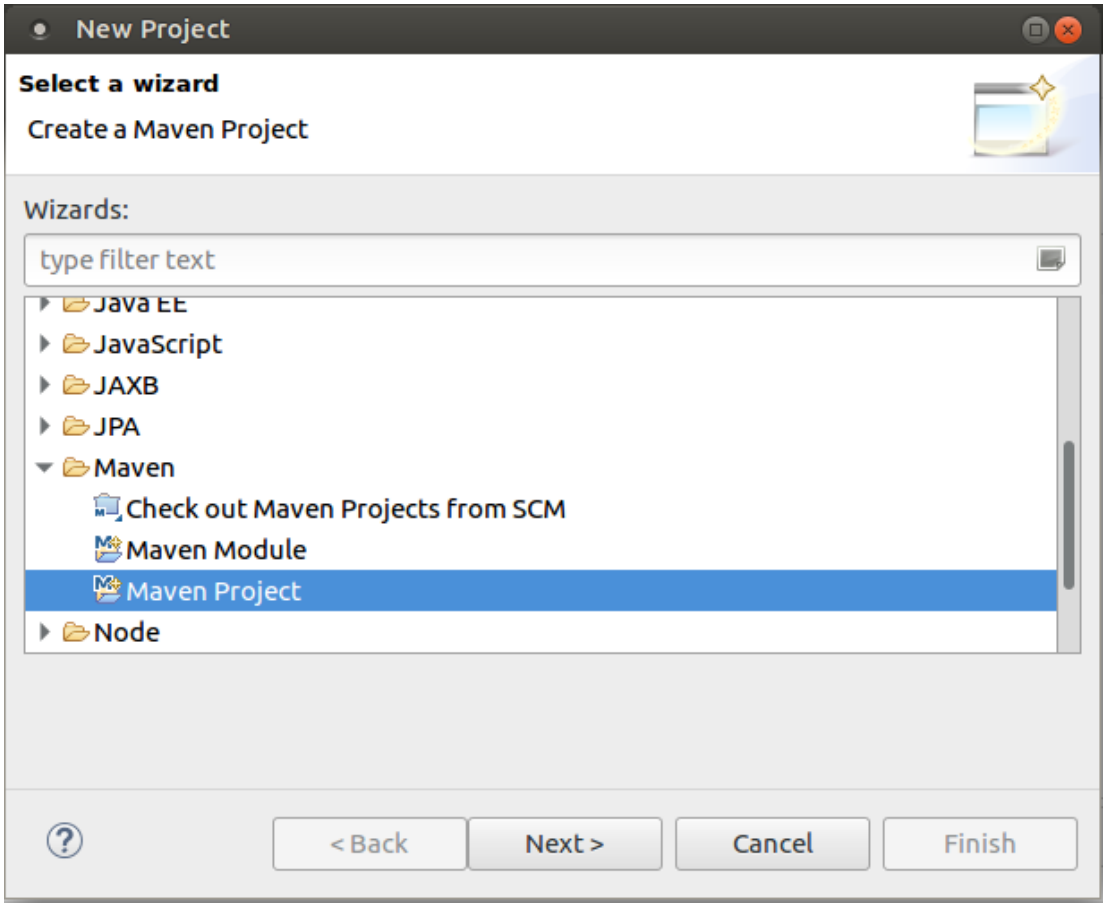


Ilustración 9 Creación proyecto Maven en Eclipse

El siguiente paso es escoger estos parámetros:

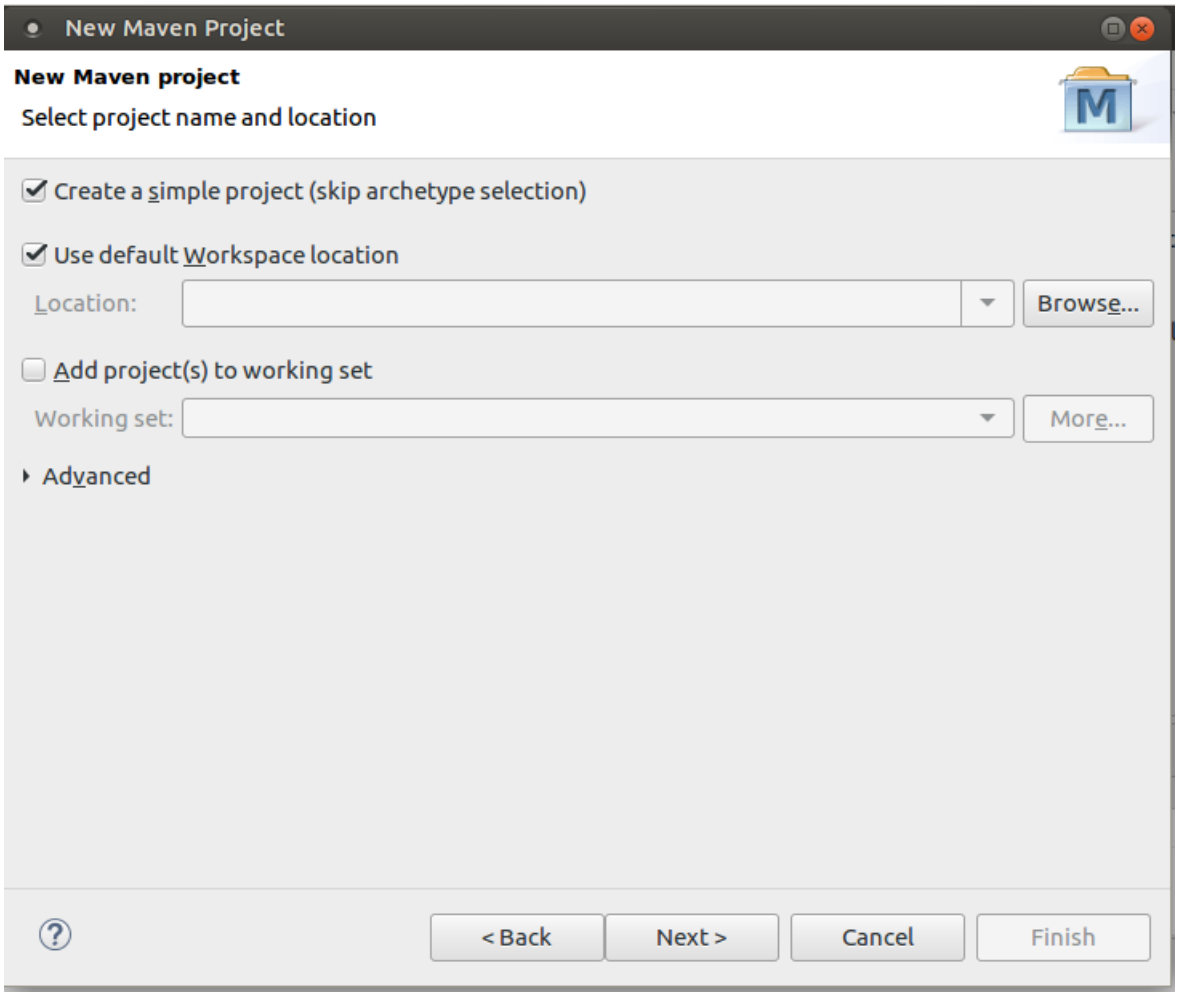


Ilustración 10 Paso 1 creación proyecto Maven

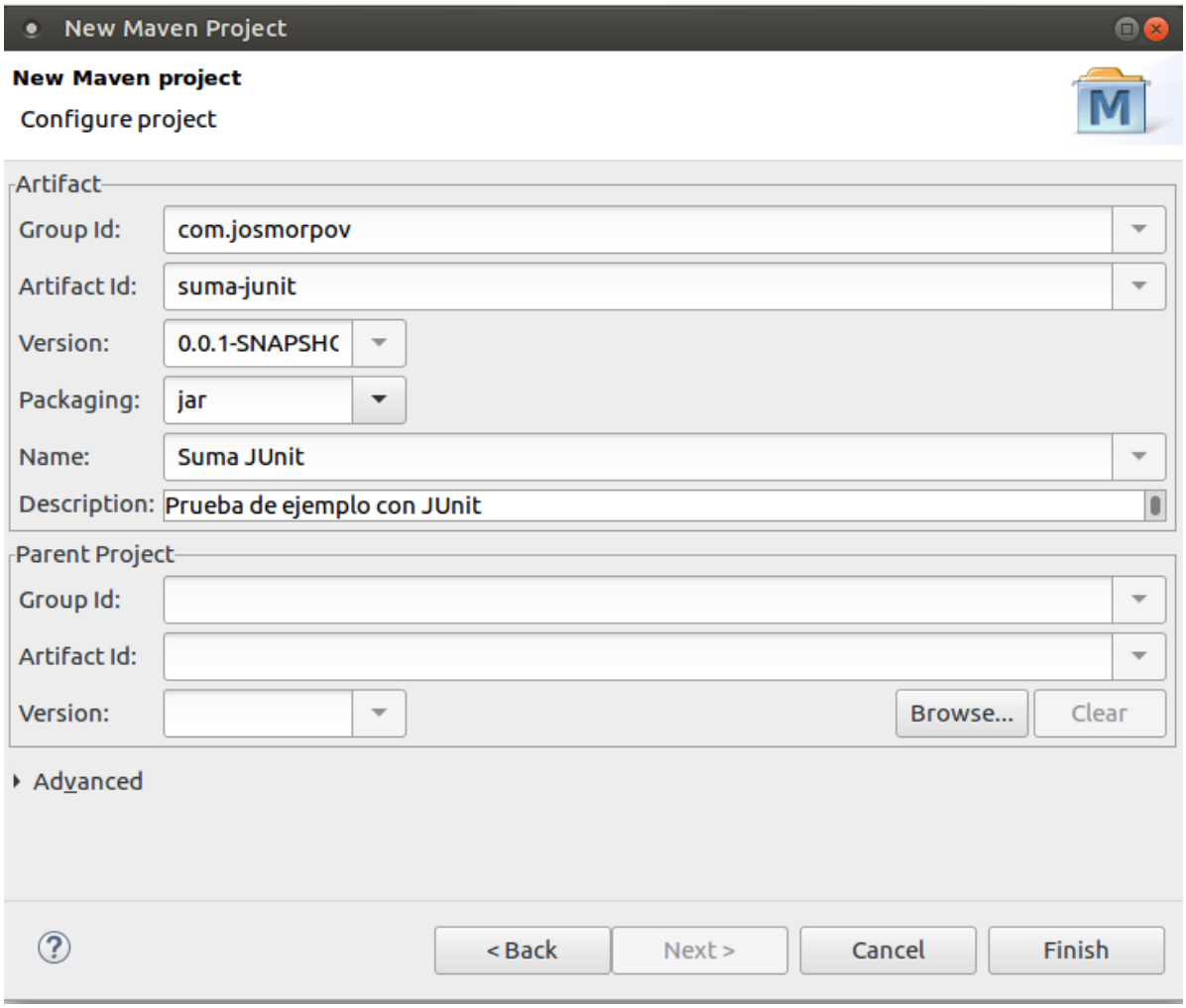


Ilustración 11 Paso 2 creación proyecto Maven

Una vez que le damos al botón de finalizar, se crea el proyecto con la siguiente estructura:

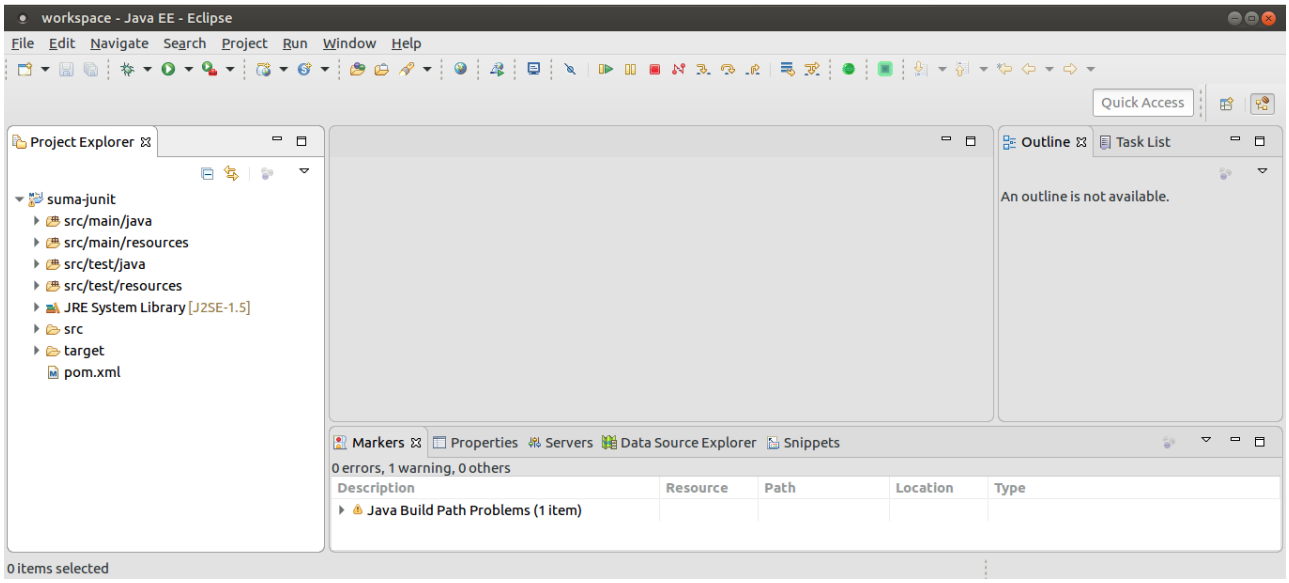


Ilustración 12 Estructura proyecto Maven

Una vez tenemos el proyecto creado, vamos a meternos en el fichero pom.xml y vamos a implementar las dependencias con JUnit. Esto lo haremos introduciendo las siguientes líneas de código después de la descripción:

```
<dependencies>
  <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
  <dependency>
<dependencies>
```

Ya se ha acabado la configuración, lo siguiente sería ponernos a implementar el método en cuestión y la prueba unitaria que queremos ejecutar contra ese método para comprobar si la implementación se ha llevado a cabo correctamente.

Para implementar el método vamos a crear la clase Sumar.java en la ruta src/main/java de nuestro proyecto y dentro de esta clase vamos a implementar el método suma que queremos comprobar, el cual recibirá como parámetros dos enteros y devolverá la suma de estos:

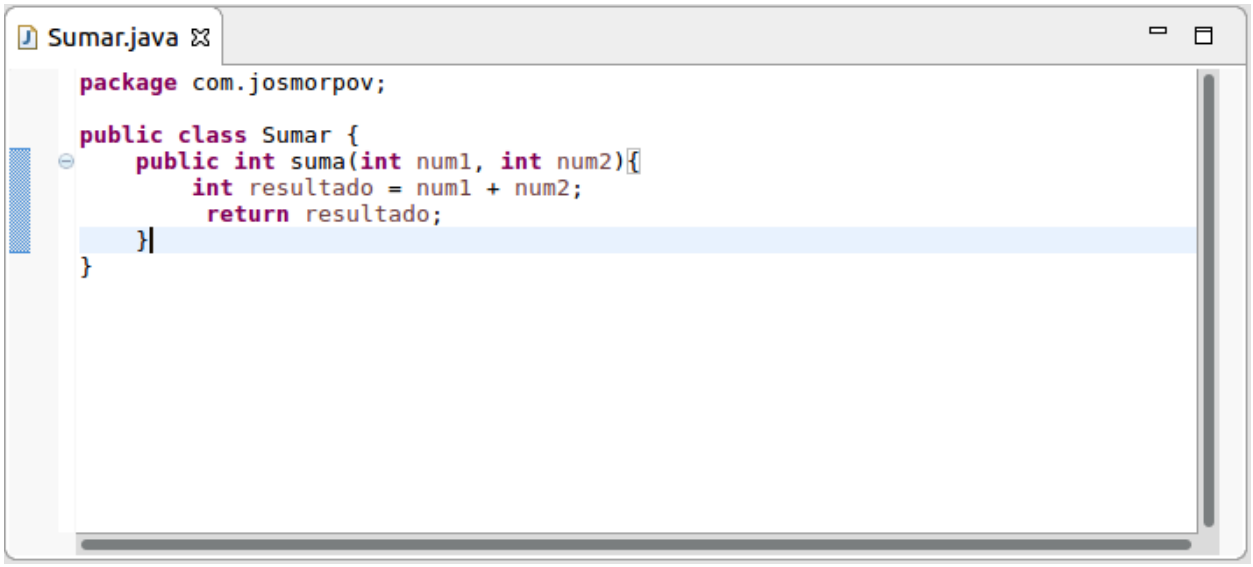


Ilustración 13 Clase Sumar

El siguiente paso una vez que tenemos nuestro método ya implementado será crear el test unitario para su comprobación. Se va a crear con el mismo nombre que la clase anterior pero añadiéndole la palabra Test al final de este. Se creará en la ruta src/test/java.

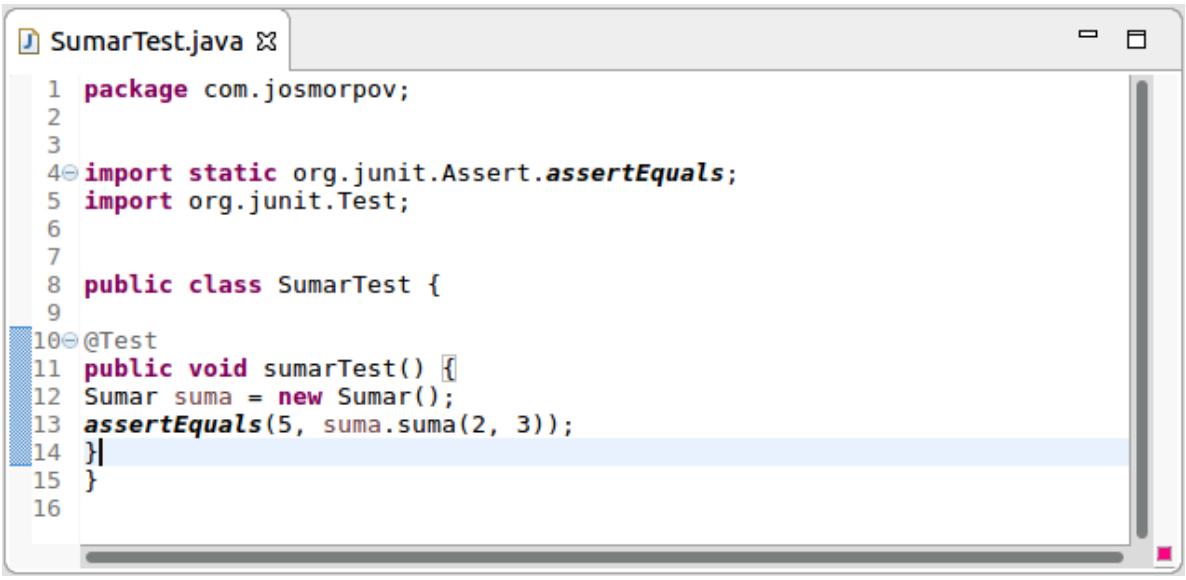


Ilustración 14 Test unitario

El test unitario que se ha creado va a comprobar que la suma 2 más 3 es igual a 5 y el test devolverá un resultado positivo o negativo dependiendo de si el método está correctamente implementado o no.

En la primera línea podemos ver el paquete con el que se ha creado el proyecto.

En las dos líneas siguientes las importaciones de bibliotecas, tanto la de Test necesaria para crear el test unitario como la biblioteca correspondiente a los assert que se implementa para hacer comprobaciones.

Después nos encontramos con una anotación @Test. Esta anotación nos indica que la siguiente función dentro del script es un test. JUnit tiene varias anotaciones que dependiendo de cuál sea harán una cosa u otra, son estas:

- **@BeforeClass**: Es usado para escribir código que se ejecutará antes de todas las pruebas.
- **@Before**: Código que se ejecutará antes de cada uno de los tests.
- **@Test**: Código de la prueba unitaria.
- **@After**: Código que se ejecutará después de cada una de las pruebas.
- **@AfterClass**: Código que se ejecutará después de todas las pruebas.

Volviendo al script, nos encontramos con la creación de un objeto Sumar el cual se va a utilizar para llamar al método que queremos comprobar y almacenar su resultado para con el assertEquals ver si se cumple la condición con la cual el test se ejecutaría con resultado positivo.

El último paso es ejecutar la prueba. El test se ejecuta haciendo clic con el botón izquierdo encima del test y eligiendo la opción Run As -> JUnit.

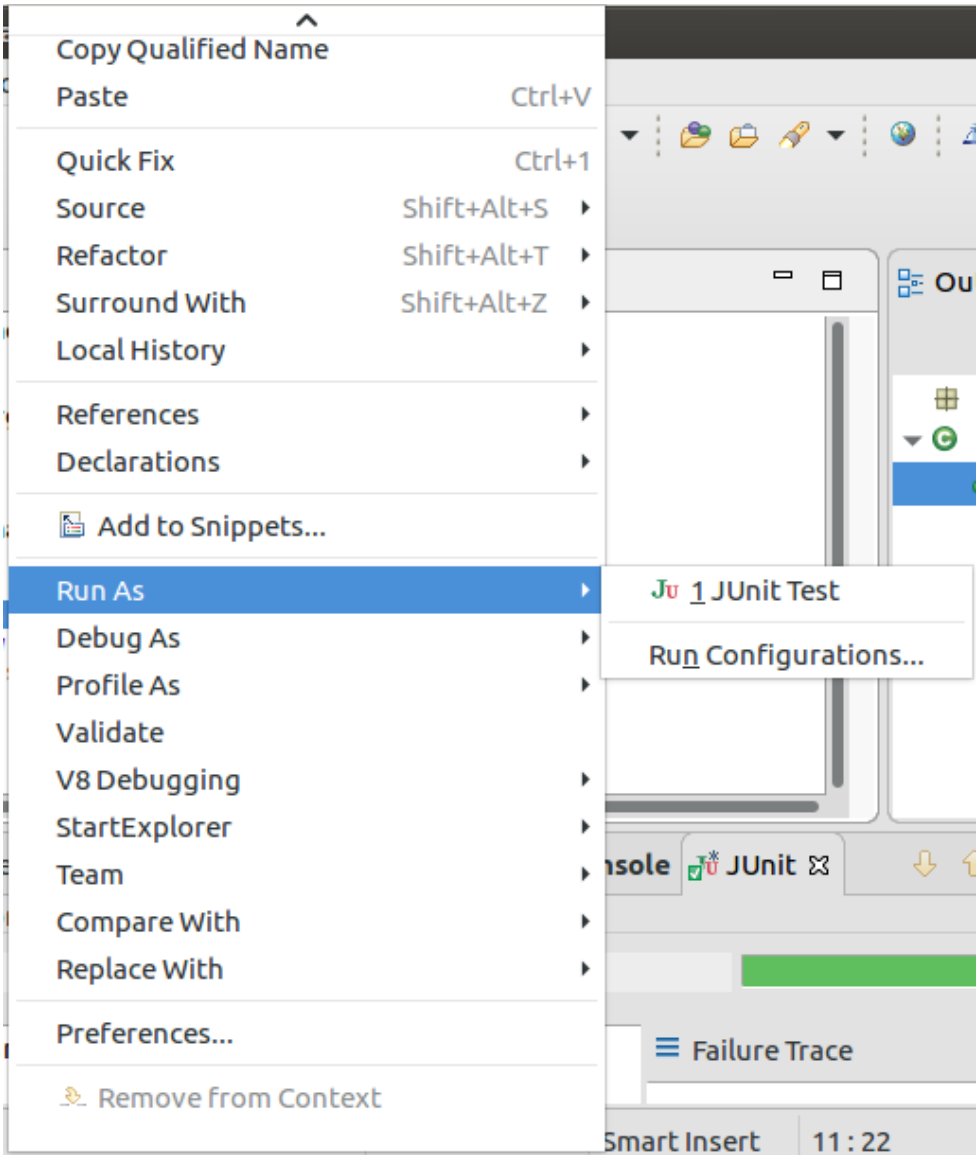


Ilustración 15 Como ejecutar un test

El resultado de esta ejecución es el siguiente:

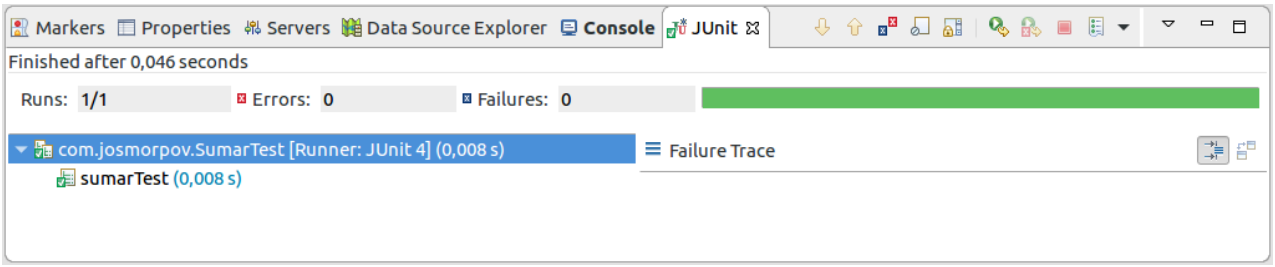


Ilustración 16 Resultado test unitario

Como podemos comprobar el resultado de este test unitario es positivo. Nos incluye el tiempo de

ejecución y en caso de ser fallido podríamos comprobar en la sección Failure Trace el fallo correspondiente.

En caso de que hubiese fallado, deberíamos de ver el porqué, refactorizar el método implementado y volver a correr el test unitario sobre dicho método.

Es por esto por lo que en muchas ocasiones se suelen crear los test unitarios antes que los propios métodos, para así tener claros los objetivos que tiene que cumplir este test y que sea más fácil su implementación.

Haciendo un pequeño inciso, para las pruebas de integración de las que hablamos en el capítulo introductorio también se pueden usar estas herramientas de las que hemos hablado. Ahora no se comprobarán métodos aislados como tal, sino que se comprobarán las diferentes integraciones de unos métodos con otros. Se puede hacer de la misma forma utilizando el algoritmo TDD nombrado, ya que conoceremos previamente el resultado esperado, con lo cual se desarrollarán estas pruebas partiendo de esta base.

Estas pruebas aumentarán sus posibles casuísticas y serán más complejas ya que se comprobarán funcionalidades conjuntas y dependientes, con lo cual será más complicado localizar y acotar los posibles fallos para una posterior refactorización.

En el siguiente capítulo empezaremos a hablar sobre las pruebas del sistema, de las cuales vamos a centrarnos en las pruebas de rendimiento y las de seguridad. Además al final del trabajo veremos por encima las pruebas de aceptación las cuales también nombramos antes junto con las unitarias, las de sistema y las de integración.

3 PRUEBAS DE SEGURIDAD

En este capítulo vamos a centrarnos nuevamente en otro tipo de pruebas, más concretamente en las pruebas de seguridad para evitar posibles vulnerabilidades y la exposición de nuestra aplicación y todo lo que se engloba en ella, como pueden ser datos de los usuarios, lo cual es información bastante sensible.

Como en el anterior capítulo, vamos a definir que son este tipo de pruebas, los diferentes tipos en los que se pueden englobar y también veremos algunas de las herramientas más usadas para la automatización de este tipo de pruebas. Además veremos un ejemplo práctico de una prueba automática con alguna de las herramientas que vamos a introducir en el penúltimo apartado del capítulo.

3.1 ¿Qué son las pruebas de seguridad?

Primeramente vamos a definir y a conocer que es la seguridad cuando se habla sobre aplicaciones software. La seguridad es un conjunto de medidas para proteger una aplicación contra acciones imprevistas que comprometen la funcionalidad de la aplicación o de los usuarios de esta. Estas acciones pueden ser o no intencionadas [8].

Hablando de seguridad, tenemos un ejemplo muy claro de la importancia que está tomando esta, y no es más que la actual ley sobre Reglamento General de Protección de Datos, más conocida como GDPR, de la que todo el mundo estamos al tanto. Entro en vigor el 25 de mayo de este año 2018. Ha implementado un nuevo marco legal en la Unión Europea para la protección y la distribución de datos personales los cuales podrían por si solos identificar a una persona.

Las aplicaciones software manejan millones de datos personales al día de sus usuarios es por esto por lo que se está volviendo cada vez más importante la seguridad y realizar este tipo de pruebas es una buena medida para evitar las pérdidas y filtraciones de este tipo de datos, lo cual podría conllevar multas de hasta 10 millones de euros o más.

Las pruebas de seguridad son una variante de las pruebas software que garantizan que el sistema y las aplicaciones que estamos testeando estén libres de cualquier posible bug que pudiera comprometer la funcionalidad de esta o que pudiera causar una gran pérdida.

Estas pruebas consisten en encontrar todas las posibles lagunas y debilidades con respecto a la seguridad de la aplicación con el fin de corregir en el menor tiempo posible dichos puntos vulnerables y construir una aplicación segura sin poner en riesgo a los usuarios.

3.2 Integración de las pruebas de seguridad con el SDLC

Primeramente decir que cuando nombramos el SDLC, nos referimos al ciclo de vida del desarrollo software. Este ciclo de desarrollo viene definido en el portal Wikipedia de la siguiente forma: "Es el proceso de creación o modificación de los sistemas, modelos y metodologías que la gente usa para desarrollar estos sistemas de software. El concepto general se refiere a la computadora o sistemas de información. En ingeniería de software el concepto de SDLC sostiene muchos tipos de metodologías de desarrollo de software. Estas metodologías constituyen el marco para la planificación y el control de la creación de una información en el proceso de desarrollo de software"

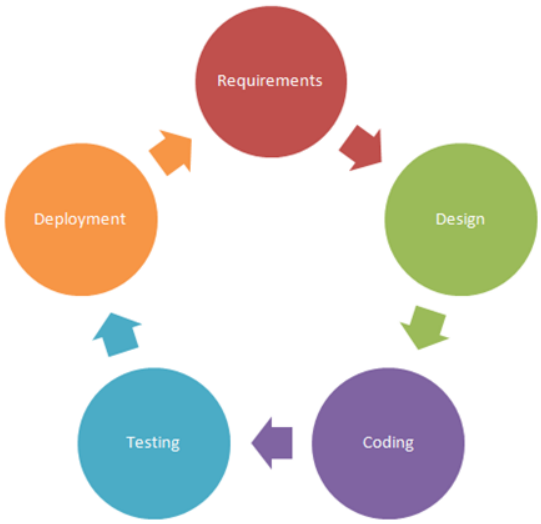


Ilustración 17 SDLC

https://www.vskills.in/certification/blog/software-development-life-cycle-sdlc/

Teniendo en cuenta lo que significa este ciclo de desarrollo y las etapas que tiene, se concuerda que si se pospusiesen dichas pruebas de seguridad hasta después del despliegue el costo será mucho mayor, por lo que es conveniente incluir este tipo de pruebas en fases anteriores.

Vamos a ver cómo sería el proceso ideal relacionando dichas pruebas de seguridad con el ciclo de vida software, para esto vamos a relacionar en una tabla los diferentes procesos del ciclo de vida de desarrollo software con los procesos de seguridad relativos a cada una de las fases de este:

Fases del ciclo de vida de desarrollo software (SDLC)	Procesos de seguridad
Requisitos	Análisis de seguridad para concretar los requisitos y verificación de casos de abuso/uso indebido.
Diseño	Análisis de los riesgos de seguridad. Desarrollo de los planes de prueba de seguridad.
Codificación y pruebas unitarias	Pruebas de caja blanca de seguridad
Pruebas de integración	Pruebas de caja negra de seguridad
Pruebas del sistema	Análisis de las pruebas de caja negra y exploración de vulnerabilidades
Implementación	Realización de pruebas de penetración.
Apoyo	Análisis de impacto de soluciones

Tabla 1 Relación SDLC con procesos de seguridad

En cuanto a esta tabla hemos visto que en la penúltima fase del SDLC se corresponden con las pruebas de penetración. Estas pruebas son simulaciones de ataques de un hacker externo a la aplicación

En esta tabla también se han hablado de pruebas de caja negra y caja blanca, las cuales no han sido explicadas. Son metodologías diferentes a la hora de realizar una prueba y se diferencian básicamente en el conocimiento o no de las entradas antes de la ejecución de una prueba. Vamos a definirlas brevemente en el contexto de las pruebas de seguridad:

1. **Caja negra:** Se realizan test de penetración donde el individuo que los ejecuta no conoce información alguna sobre el sistema. Refleja de la mejor manera posible un ataque real de un hacker externo a la compañía y a la aplicación software.

2. **Caja blanca:** Se realizan test de penetración donde el tester conoce toda la información acerca del sistema. Entre esta información podemos destacar los mapas de red, accesos, código fuente…

3. **Caja gris:** Como nos podemos imaginar por su nombre es una mezcla entre las dos anteriores, la metodología de caja negra y la de caja blanca. Aquí el tester cuenta con información pero esta es información limitada. La función principal de esta metodología es profundizar en aquellos puntos críticos.

3.3 Herramientas utilizadas

En este tercer apartado vamos a hacer como en el anterior capítulo, vamos a hacer referencia a las herramientas más conocidas para la automatización de pruebas de seguridad. Vamos a ir nombrándolas y definiéndolas una a una y viendo sus diferentes características.

3.3.1 OWASP ZAP

Es una de las herramientas para la realización de pruebas de seguridad más populares. Es gratuita y de libre distribución (Open Source). Fácil de usar y tiene como finalidad encontrar vulnerabilidades y posibles fallos de seguridad en una aplicación web.

Se puede utilizar como servidor proxy, con lo que permitirá al usuario manipular todo el tráfico que pasa por este, incluido el de URL's de tipo https.

Está desarrollada en el lenguaje de programación Java. Es una herramienta multiplataforma disponible para la mayoría de Sistemas Operativos, Windows, Linux, Mac OS X…

Ilustración 18 OWASP ZAP

https://www.owasp.org

Está diseñada para ser utilizada tanto por desarrolladores y testers o personas con poca experiencia en tests intrusivos como para personas que tengan un alto conocimiento en este tipo de pruebas.

Esta herramienta permite automatizar las pruebas y también facilita un número de herramientas para hacerlas manualmente.

Sus principales funcionalidades son:

- Posibilidad de actuar como un proxy, con lo que nos permitirá ver las solicitudes que se hacen a una aplicación web y las respuestas que de esta se reciben.

- Intercepción de llamadas AJAX.

- Escáneres automatizados y pasivos.
- Fuzzer, es capaz de generar y enviar datos secuenciales o aleatorios con el objetivo de localizar fallos.
- Da soporte para un gran número de lenguajes de scripting (ECMAScript, Groovy, Phyton…).
- Soporte de WebSocket, que es una tecnología que nos proporciona sobre un único socket TCP un canal de comunicación bidireccional y full dúplex [9].
- Potente API (Application Programming Interfaces) basada en REST, el cual es un estilo de arquitectura software.

Se podrían nombrar más pero estas son las principales ventajas que tiene esta herramienta para pruebas de seguridad.

3.3.2 ScanMyServer

ScanMyServer proporciona uno de los informes más completos en cuanto a pruebas de seguridad se refiere. Se incluyen pruebas como:

- Inyección SQL, es un método mediante el cual se realiza una infiltración de código intruso aprovechando una vulnerabilidad de la aplicación en el nivel de validación de las entradas para realizar operaciones sobre una base de datos.
- Cross Site Scripting (XSS), es otro tipo de vulnerabilidad informática presente en aplicaciones web, mediante las cuales se puede inyectar código JavaScript o de un lenguaje similar. XSS es un vector de ataque que puede ser utilizado para robar información delicada, secuestrar sesiones de usuario…
- Inyección PHP, de la misma forma que se puede inyectar código SQL, HTML…se puede inyectar código malicioso PHP con la misma finalidad.
- Inyección de encabezado HTTP (Protocolo de transferencia de hipertexto), la inyección de estas cabeceras puede permitir la división de respuestas HTTP, fijación de sesión a través de las cookies, XSS, ataques de redireccionamiento…

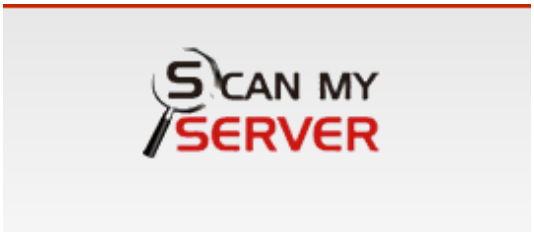


Ilustración 19 ScanMyServer

https://sensorstechforum.com/6-cyber-security-services-protect/

Otra de las ventajas es que se trata de un software abierto. Únicamente tienes que introducir la URL de tu aplicación web y nos llegará el debido informe por correo electrónico.

Ilustración 20 ScanMyServer URL

https://scanmyserver.com

3.3.3 HP WebInspect

HP WebInspect es una herramienta de pruebas de penetración y seguridad de las aplicaciones web.

Esta herramienta permite realizar pruebas de penetración y seguridad de las aplicaciones web de forma automatizada y configurable, imitando las técnicas y ataques de piratería del mundo real, permitiendo analizar detalladamente la aplicación web en busca de vulnerabilidades de seguridad.

HP WebInspect permite probar nuestra aplicación desde el desarrollo hasta la producción, gestionar eficazmente los resultados de las pruebas y distribuir los conocimientos de seguridad aprendidos.

Ilustración 21 HP WenInspect

https://www.tecnogaming.com/2013/05/hp-ayuda-a-las-organizaciones-a-identificar-vulnerabilidades/

Las características principales de esta herramienta son:

- Pruebas de seguridad de las aplicaciones web desde el desarrollo hasta la producción.
- API web y servicios web de pruebas de seguridad.
- Gestión sencilla y con la opción de compartir fácilmente resultados e historiales de pruebas de seguridad.
- Permite adoptar cualquier ciclo de vida para hacer cualquier tipo de pruebas de seguridad automáticas dependiendo de la fase en la que nos encontremos.
- Comprueba la conformidad con las regulaciones y las políticas de seguridad actuales.

3.4 Ejemplo de prueba de seguridad

En la actualidad mi trabajo en este campo se ha centrado en la herramienta OWASP ZAP, la cual la tenemos configurada en nuestro entorno de trabajo para cada vez que se lance una prueba funcional contra la aplicación, de la que hoy en día estamos encargados, se lance previamente la ejecución de este software de seguridad para detectar los posibles fallos y vulnerabilidades del sistema para contrarrestarlos lo más rápido posible y hacer de esta una aplicación segura y estable.

Estos datos son datos bastante sensibles y no se pueden compartir fuera de las organizaciones si no, no serían pruebas de seguridad y los datos estarían al alcance de cualquiera. Por esta razón vamos a coger esta herramienta y hacer una prueba de seguridad sencilla sobre la aplicación del Aula Virtual de la Escuela de Ingenieros: www.ev.us.es

Como dijimos anteriormente cuando vimos OWASP ZAP, es muy sencilla de utilizar ya que básicamente las funciones básicas se limitan a coger la URL del portal sobre el que queremos lanzar dichas pruebas, introducirlas en un campo destinado a ello sobre la interfaz de dicha herramienta y darle a un botón para realizar el escaneo.

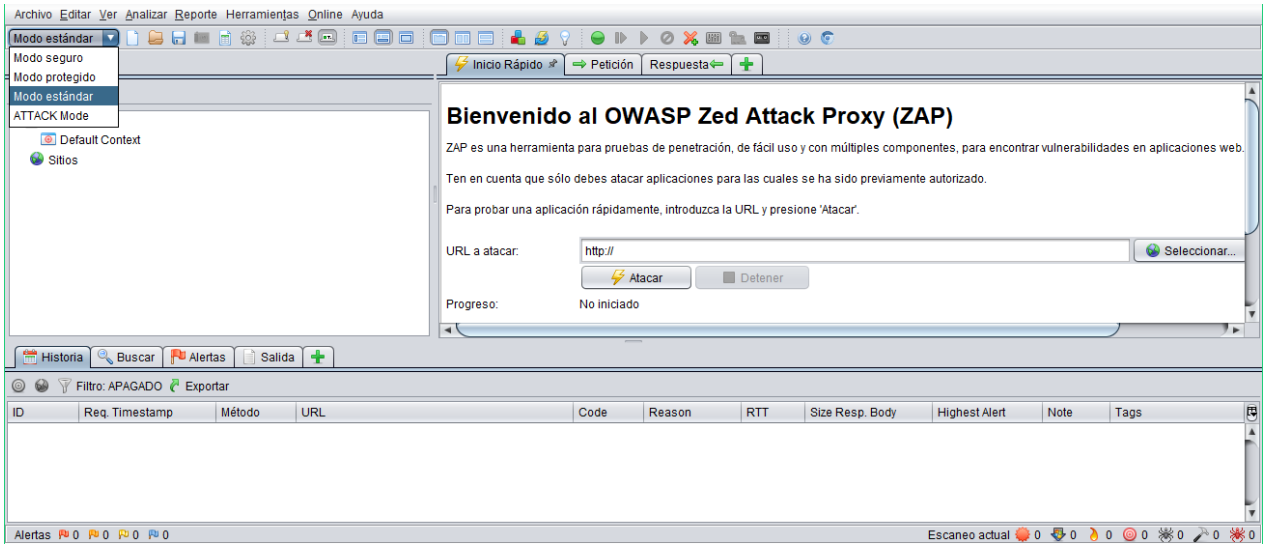


Ilustración 22 Interfaz de usuario OWASP ZAP

Podemos ver que en la parte superior tenemos la típica barra de herramientas para llevar a cabo funciones básicas de manera rápida. Dentro de esta barra de herramientas podemos ver un desplegable con los diferentes modos en los que se puede realizar un análisis de seguridad dependiendo de las necesidades que tengamos.

Justo en el panel inferior al desplegable nos encontramos con una sección en la que se van a ir almacenando todos los portales web que vayamos visitando.

A su derecha tenemos la sección para introducir la URL de la aplicación web y los botones para iniciar el análisis o pararlo.

En la sección más inferior nos encontraremos la parte donde saldrá toda la información relativa a los análisis realizados.

Vamos a introducir la URL antes mencionada y posteriormente haremos un análisis de los resultados

obtenidos.

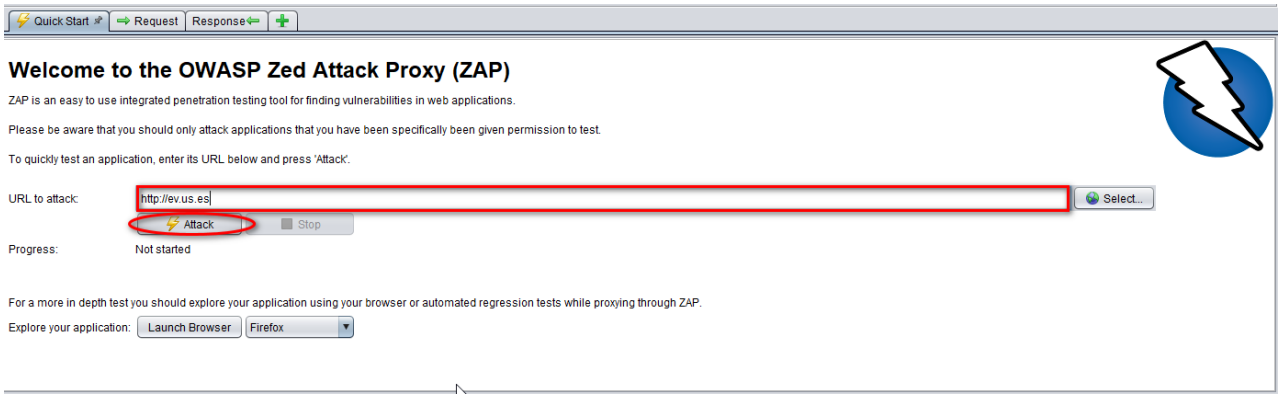


Ilustración 23 Introducción de URL para su análisis

Una vez que tenemos introducida la URL y le damos al botón Attack, empieza a realizar el escaneo.

Una vez realizado este escaneo, los resultados se almacenan en diferentes grupos depende de la acción realizada. Vamos a ver los diferentes tipos de resultados que arroja el escaneo de dicha aplicación:

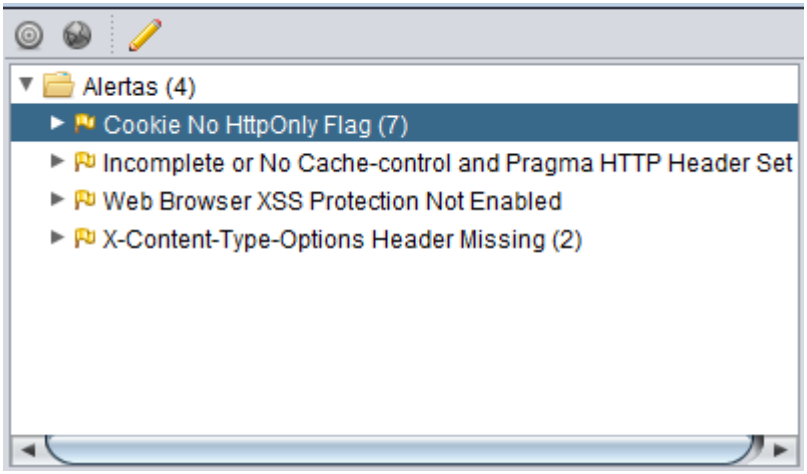


Ilustración 24 Diferentes tipos de alertas como resultado

Si nos metemos en el tercer tipo, Web Browser XSS Protection Not Enabled, encontramos una incidencia. Seleccionando la incidencia desplegada, podemos ver lo siguiente:

Ilustración 25 Protección XSS inhabilitada

Como hemos visto anteriormente, Cross Site Scripting (XSS) era la posibilidad de inyectar código JavaScript o de un lenguaje similar con el objetivo de robar información delicada. Aquí podemos ver que no está activo y que el riesgo dado por la aplicación es bajo.

Más abajo nos encontramos la descripción que viene a decir lo siguiente: “Web Browser XSS Protection is not enabled, or is disabled by the configuration of the 'X-XSS-Protection' HTTP response header on the web server”. Básicamente nos está dando a conocer que la protección XSS está deshabilitada para este portal web.

Esta herramienta nos proporciona posibles soluciones por cada alerta de seguridad encontrada, en este caso nos dice lo siguiente: “Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'”. Nos viene a decir que nos aseguremos de habilitar esta protección en la configuración seteando la cabecera de respuesta X-XSS-Protection HTTP a 1.

4 PRUEBAS DE RENDIMIENTO

Este cuarto capítulo va a tratar otro de los campos importantes en cuanto a pruebas automáticas sobre aplicaciones software se refiere, el de las pruebas de rendimiento. Se verá qué son este tipo de pruebas, cuál es el objetivo que perseguimos a la hora de ejecutar dichas pruebas sobre nuestra aplicación, una clasificación en tipos, la metodología que se sigue y como en los demás apartados se verán herramientas que nos facilitarán esa tarea de automatización. En el último punto se verán un conjunto de pruebas prácticas que el lector podrá utilizar para guiarse.

4.1 ¿Qué son las pruebas de rendimiento?

Este tipo de pruebas se ejecutan sobre las aplicaciones software para garantizar que estas funcionen correctamente según una situación y unas condiciones particulares. Se centra en la velocidad de respuesta con la que el sistema realiza una determinada tarea impuesta por una prueba en unas condiciones particulares, como por ejemplo con una carga dada, un número de usuarios conectados a la vez…

Este tipo de pruebas ayudan a detectar posibles fallos de rendimiento en diversas situaciones como cuellos de botella, antes de ser detectados por los usuarios en producción y que suponga un mal mayor que conlleve pérdidas económicas.

Como estamos viendo en este trabajo de fin de grado no todo en el mundo del testing es en referencia a las pruebas funcionales también hay que medir y tener en cuenta otros parámetros para hacer una aplicación de calidad como pueden ser su tiempo de respuesta, confiabilidad, uso de recursos y escalabilidad.

Ilustración 26 Objetivos de las pruebas de rendimiento

http://vyvquality.com/pruebas-rendimiento/

Con estos parámetros medidos determinaremos si la aplicación responde de forma rápida, la carga máxima de usuarios que esta aplicación puede soportar, si la aplicación es estable bajo demandas y cargas variables…

También es aplicable a otros aspectos diferentes a la comprobación del rendimiento de la aplicación como la investigación, la medición, la validación o la verificación de otros atributos de calidad de la aplicación, como la escalabilidad, la seguridad y el uso de recursos.

4.2 Tipos de pruebas de rendimiento

En este siguiente apartado vamos a hacer una clasificación de los diferentes tipos de pruebas de rendimiento más conocidos y más usados en el proceso de testeo de una aplicación software. Vamos a nombrarlos y a hablar un poco por encima de ellos:

- **Pruebas de carga**: Este tipo de pruebas intentará validar el alcance de los objetivos propuestos en cuanto a prestaciones de la aplicación se refiere. Trata de validar la correcta funcionalidad de la aplicación basándose en los objetivos mínimos a cumplir dados por el cliente o por nosotros mismos en caso de que sea una aplicación propia, todo esto en referencia al rendimiento. Además tratará de localizar los posibles cuellos de botella y con ello los fallos que se puedan presentar en la aplicación a lo largo de este tipo de pruebas, para poder anticiparnos y solucionar esto lo más rápido posible antes de publicar una versión de la aplicación.

- **Pruebas de estrés:** El objetivo de este tipo de pruebas es ver los límites de nuestra aplicación. Para ello las condiciones bajo las que se ejecutan dichas pruebas en la aplicación deben de ser extremas para ver cómo se desenvuelve esta con cargas y un volumen de datos bastante alto. Esto se consigue sometiendo a la aplicación a una carga por encima de los límites requeridos para su correcto funcionamiento, es decir, a una mayor carga que en el tipo de pruebas antes mencionado.

- **Pruebas de resistencia:** Este tipo de pruebas como su nombre indica sirven para verificar la resistencia de una aplicación ante una carga normal y esperada durante un periodo de tiempo prolongado. También se llaman pruebas de estabilidad. Es útil para comprobar la degradación del servicio ante un uso prolongado del sistema. Sirven para detectar fugas de memoria o problemas de corrupción.

- **Pruebas de picos:** También se le conoce a este tipo de pruebas como pruebas de Spike. En este tipo de pruebas se mide la reacción de la aplicación ante picos de carga generados de forma repentina.

- **Pruebas de escalabilidad:** Este tipo de pruebas tiene como objetivo comprobar la escalabilidad de una aplicación. Esta escalabilidad hace referencia a un posible aumento en los objetivos mínimos de rendimiento que se plantearon en las pruebas de carga. Con esto se pretende conseguir una aplicación más estable y cada vez más potente dentro de los recursos de los que disponemos.

4.3 Metodología a seguir

Aquí se va a hablar del proceso que se recomienda seguir desde que se conoce el entorno donde se van a realizar este tipo de pruebas hasta la posterior ejecución y análisis de la información y resultados obtenidos [10].

Ilustración 27 Pasos de la metodología

https://www.guru99.com/performance-testing.html

1. **Identificar el entorno de pruebas**: El primer paso a dar en este proceso será el de identificar los diferentes entornos de los que disponemos en especial el entorno de pruebas y el de producción. Además de esto es necesario conocer las herramientas y los recursos de los que disponemos el equipo de pruebas. Se incluye el hardware, software y la configuración de la red.

   Esto es así ya que tener un amplio conocimiento sobre el entorno nos va a ayudar a la planificación y el desarrollo de casos de prueba. Además de esto nos ayudará a identificar problemas en las fases tempranas del proyecto.

2. **Determinar los criterios de aceptación en cuanto a rendimiento**: Consiste en determinar los parámetros anteriores descritos, velocidad en la respuesta, estabilidad, escalabilidad, limitaciones…Esto se hace con el objetivo de medir estos parámetros y hacer un análisis para comprobar si el funcionamiento de la aplicación será el correcto.

3. **Planificación y diseño de pruebas:** En este paso es donde se tienen que identificar las diferentes casuísticas y plantear los diferentes escenarios de pruebas. Aquí se plantean los pasos a seguir, los datos que intervienen en las pruebas, las precondiciones…

4. **Configuración del entorno de prueba:** Aquí se va a realizar la preparación del entorno de pruebas antes de la ejecución de las pruebas diseñadas en el paso anterior. También se tienen que preparar las diferentes herramientas y los posibles recursos necesarios.

5. **Implementar el diseño de la prueba:** En este quinto paso lo que se hace no es más que plasmar lo definido en el punto 3 de diseño y planificación de pruebas.

6. **Ejecución de las pruebas:** Como su propio nombre indica en este paso se lleva a cabo la ejecución de las pruebas una vez diseñadas e implementadas, con el objetivo de recoger unos

resultados que se valorarán posteriormente.

7. **Análisis de resultados, creación de informes y volver a ejecutar:** El último paso consiste en analizar los datos arrojados en el paso anterior de ejecución de pruebas. Se comprobarán los valores obtenidos para ver si el funcionamiento cumple con los criterios de aceptación o no, realizar un informe con toda esta información y en caso de que sea necesario volver a ejecutar determinadas pruebas cuando los informes hayan salido negativos.

4.4 Herramientas utilizadas

Estas herramientas nos van a facilitar la realización de pruebas de rendimiento realistas, ya que sería una tarea casi imposible el tener a 10.000 usuarios, por decir una cantidad, que se conecten simultáneamente a una aplicación web y realicen una serie de tareas. En portales como Google este número de usuarios conectados simultáneamente sería muchísimo mayor, con lo cual se dificultaría aún más la tarea. Es por este motivo por el que las herramientas de automatización de pruebas de rendimiento nos van a ser muy útiles, ya que podemos realizar estas funciones sin apenas dificultad.

4.4.1 Apache JMeter

Esta aplicación es totalmente gratuita (Open Source) diseñada en primer lugar por Stefano Mazzocchi de la Apache Software Foundation. Es una aplicación puramente diseñada en Java para medir el rendimiento de aplicaciones software. Inicialmente se diseñó para medir el rendimiento en aplicaciones web y aplicaciones FTP, pero actualmente sus funciones se han extendido más allá de estas como en las pruebas de servidores de bases de datos.

Ilustración 28 JMeter

https://octoperf.com/blog/2018/03/29/jmeter-tutorial/

Vamos a ver las principales ventajas de este framework de pruebas de rendimiento automáticas en un gráfico y vamos a ir explicando por encima en que consiste cada una [11]:

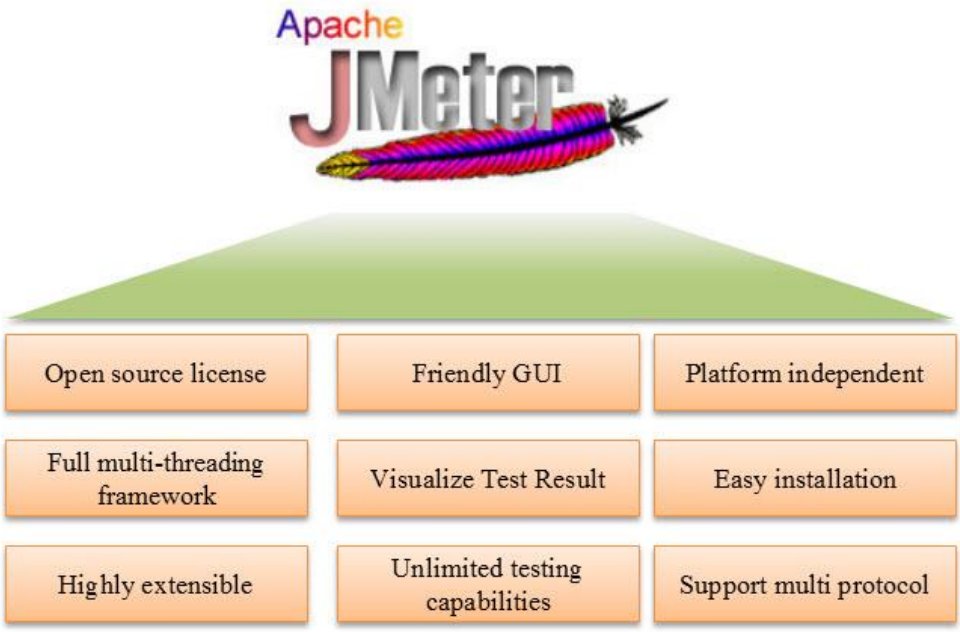


Ilustración 29 Ventajas JMeter

https://www.guru99.com/introduction-to-jmeter.html

- Es un software gratuito (Open Source), lo que significa que es totalmente gratuito y de libre distribución.
- Es una aplicación sencilla de usar y lleva poco tiempo familiarizarse con ella.
- Plataforma independiente: Es una aplicación de escritorio 100% Java, por lo que puede ejecutarse en múltiples plataformas.
- Framework multi-hilo, lo cual permite realizar distintas funciones de forma simultánea en hilos separados.
- Los informes de los resultados de las pruebas son bastante flexibles y orientativos, ya que se pueden mostrar en formatos muy diferentes.
- Instalación sencilla.
- Es extensible en cuanto a instalación de complementos y diferentes plugins que nos aportan nuevas funcionalidades.
- Admite muchos tipos de pruebas diferentes con lo cual lo hace un software bastante completo.
- Soporte multiprotocolo, ya que JMeter aunque en un principio fue diseñado para plataformas web, actualmente ha crecido y evalúa otro tipo de funcionalidades y plataformas. Protocolos como HTTP, JDBC, LDAP, SOAP, JMS y FTP son compatibles.

- Se puede integrar con otras herramientas de automatización de pruebas como Selenium del que hablaremos más tarde para la realización de pruebas automáticas.

4.4.2 HP Loadrunner

Es una de las herramientas más utilizadas y populares hoy en día en cuanto a pruebas de rendimiento se refiere. Fue creada por la multinacional inglesa Micro Focus.

La funcionalidad más usada de esta aplicación es la posibilidad de similar cientos de miles de usuarios que se conectan a una determinada aplicación software y simulan acciones de la vida real.

Ilustración 30 HP Loadrunner

https://en.wikipedia.org/wiki/LoadRunner

Esta se compone a su vez de cuatro herramientas [12]:

- Virtual User Generator (Vugen), es una herramienta encargada de simular el comportamiento de usuarios reales mediante la creación de scripts automáticos.
- Controller, tiene las funcionalidades de organizar, conducir, administrar y supervisar las pruebas de carga vistas con anterioridad.
- Load Generators (Generadores de Carga), su función principal es la de generar la carga que será aplicada a la aplicación software.
- Analysis, que como su propio nombre indica es una herramienta cuyo fin es el de analizar y comparar los resultados de las pruebas.

Esta herramienta incluye soporte para:

- Aplicaciones que usan Microsoft .NET y Java.
- Servidores de bases de datos como Microsoft SQL Server y Oracle.
- Protocolos de internetworking como DNS , FTP y LDAP.
- Protocolos de correo electrónico que incluyen IMAP , MAPI , POP3 y SMTP.

- Tecnologías de cliente remoto como Citrix ICA y RDP.

4.4.3 WebLOAD

Esta herramienta está cogiendo mucha fuerza en la actualidad, reemplazando en muchos casos a la anterior expuesta Loadrunner. Fue creada por la empresa RadView Software.

Ilustración 31 WebLOAD

https://en.wikipedia.org/wiki/WebLOAD

Es una herramienta del mismo estilo que las anteriores, cuya funcionalidad principal no es otra que la automatización de pruebas de carga. Esta tiene ventajas con respecto a la anteriormente nombrada en cuanto a que puede llegar a duplicar la carga de usuarios virtuales en una aplicación web. Su costo también es una ventaja a destacar, ya que es una herramienta bastante más barata de adquirir. También podemos decir sobre esta que es una herramienta mucho más rápida que sus competidoras.

Otra ventaja a destacar es la posible integración de estas herramientas con otros frameworks de pruebas como por ejemplo Selenium, en caso de automatización de pruebas funcionales para aplicaciones web, y Appium, en caso de automatización de pruebas funcionales esta vez para aplicaciones móviles.

Como última ventaja diremos que es compatible con un número elevado de protocolos y diferentes tecnologías web, móviles y empresariales. En el siguiente link de su página oficial se pueden consultar todas las tecnologías, protocolos y demás para los que es compatible:

https://www.radview.com/technologies-supported/

4.4.4 Google PageSpeed Insights

Es una herramienta totalmente gratuita creada por Google, la cual tiene como función principal medir el rendimiento de aplicaciones web tanto en dispositivos móviles como en ordenadores.

Aparte de esto una vez analiza estas características y arroja un resultado, nos propone mejoras a implementar en nuestras aplicaciones para mejorar este tema del rendimiento.

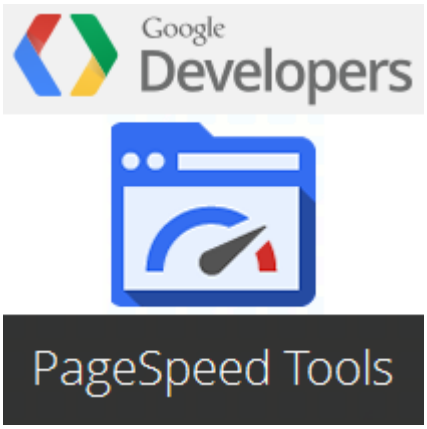


Ilustración 32 Google PageSpeed Insights

https://insideout.com/blog/2013/11/20/website-load-speed-google-pagespeed-insights/

Se puede utilizar esta herramienta como servicio online y de forma muy sencilla, únicamente nos bastará con introducir la URL del portal que queremos evaluar.

Los objetivos de estas buenas prácticas que nos ofrece Google son, principalmente:

- Minimizar el número de peticiones HTTP que se producen.
- Reducir el tamaño de las respuestas de estas peticiones HTTP.
- Optimizar el renderizado de la página en el navegador.

PageSpeed analizará nuestra aplicación web y nos asignará una puntuación o score sobre 100 que evalúa cuánto más rápida podría ser la carga de esta web. Un score muy alto y cercano a 100 indica que hay poco que mejorar (es decir, la aplicación ya carga todo lo rápido que puede o cerca) mientras que un score bajo indica que hay bastantes mejoras que podríamos llevar a cabo.

Además de la herramienta online, podemos encontrar una extensión totalmente gratuita (Open Source) para Chrome Developer Tools (un conjunto de herramientas para desarrolladores integradas dentro del navegador Google Chrome) y también para Firebug (una extensión gratuita para el navegador Mozilla Firefox con el cual se facilitan las tareas de depuración, edición y monitorización de cualquier sitio web) para los usuarios del navegador Firefox.

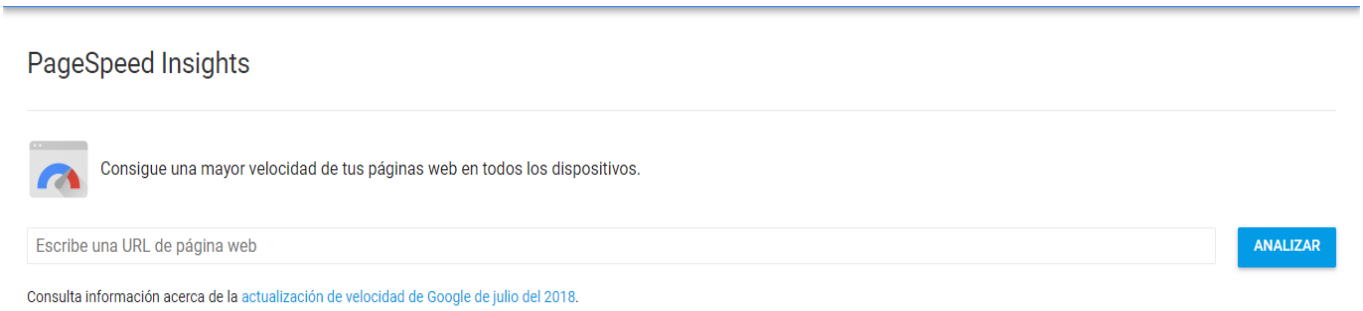


Ilustración 33 Captura de la herramienta PageSpeed Insights

https://developers.google.com/speed/pagespeed/insights/?hl=es

4.5 Ejemplo de prueba de rendimiento

Para este ejemplo práctico sobre la realización de pruebas de rendimiento de forma automática he cogido el portal web para el que actualmente estoy trabajando. Es un portal dedicado al juego online en España, llamado Wanabet.

Tenemos diferentes entornos de trabajo como es normal en compañías grandes. En este caso contamos con nuestro entorno local en el que los desarrolladores implementan su código y le hacen las pruebas unitarias pertinentes, un entorno de desarrollo propiamente dicho donde se hacen las primeras pruebas tanto manuales como automáticas, un entorno de preproducción muy similar al de producción pero sin datos sensibles de los usuarios de la plataforma y por último el entorno de producción que es el entorno con el que un usuario exterior puede interactuar.

El entorno escogido para estas pruebas va a ser el de desarrollo ya que es un entorno en el que el impacto de realizar cualquier acción es mucho menor que en cualquier otro. Nos dará unos resultados próximos a la realidad aunque el volumen de datos que se procese en estos entornos sea mucho menos que en uno de producción. Además, estos resultados también dependerán de la máquina en la que estemos lanzando dichas pruebas.

La herramienta escogida es JMeter, ya que es una de las herramientas más completas como vimos en el punto anterior y es muy configurable, es la herramienta que utilizamos día a día los tester de la compañía de R. Franco Digital para realizar las pruebas de rendimiento sobre nuestros portales.

4.5.1 Descarga y primeros pasos en JMeter

Lo primero que deberíamos hacer es descargarnos JMeter desde su página oficial:

https://jmeter.apache.org/download_jmeter.cgi

Una vez descargada la herramienta nos quedará una carpeta con el nombre del framework y la versión. Nos moveremos a la ruta apache-jmeter-5.1.1\bin y allí vamos a encontrar un ejecutable Jar llamado Apache JMeter, el cual ejecutaremos y se nos abrirá la ventana de la herramienta.

examples	10/03/2019 8:36	Carpeta de archivos	
report-template	10/03/2019 8:36	Carpeta de archivos	
templates	10/03/2019 8:36	Carpeta de archivos	
ApacheJMeter	10/03/2019 9:58	Executable Jar File	14 KB
BeanShellAssertion.bshrc	10/03/2019 10:08	Archivo BSHRC	2 KB
BeanShellFunction.bshrc	10/03/2019 10:08	Archivo BSHRC	3 KB
BeanShellListeners.bshrc	10/03/2019 10:08	Archivo BSHRC	2 KB
BeanShellSampler.bshrc	10/03/2019 10:08	Archivo BSHRC	3 KB
create-rmi-keystore	10/03/2019 8:36	Archivo por lotes ...	2 KB
create-rmi-keystore.sh	10/03/2019 8:36	Archivo SH	2 KB
hc.parameters	10/03/2019 10:08	Archivo PARAMET...	2 KB
heapdump	10/03/2019 8:36	Script de comand...	2 KB
heapdump.sh	10/03/2019 8:36	Archivo SH	2 KB
jaas.conf	10/03/2019 10:08	Archivo CONF	2 KB
jmeter	10/03/2019 8:43	Archivo	8 KB
jmeter	10/03/2019 8:43	Archivo por lotes ...	9 KB
jmeter.properties	10/03/2019 10:08	Archivo PROPERTI...	56 KB
jmeter.sh	10/03/2019 8:36	Archivo SH	4 KB
jmeter-n	10/03/2019 8:43	Script de comand...	2 KB
jmeter-n-r	10/03/2019 8:36	Script de comand...	2 KB
jmeter-server	10/03/2019 8:43	Archivo	2 KB

Ilustración 34 Carpeta bin JMeter

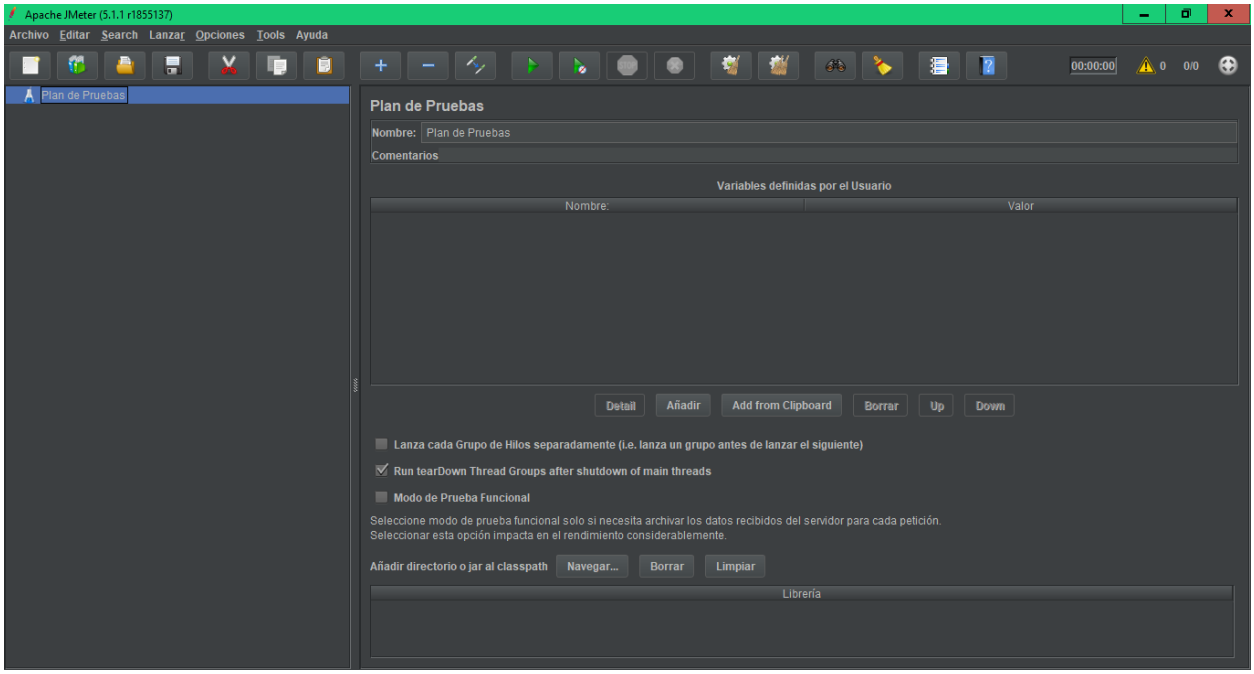


Ilustración 35 Frontal aplicación JMeter

Antes de meternos a utilizar JMeter, para realizar dichas pruebas tenemos que hacer previamente un estudio sobre los flujos a los cuales queremos pasarle estas pruebas de rendimiento. Al tratarse de pruebas costosas en cuanto a tiempo y otros recursos, debemos definir unos flujos prioritarios con los cuales se cumplirían las principales funcionalidades de nuestra aplicación.

Al tratarse de una aplicación de juego online, podemos definir los siguientes flujos como prioritarios:

registrarse, loguearse, depositar dinero para jugar, jugar y retirar dinero.

Una vez definidos estos flujos debemos de conocer las diferentes llamadas a nuestras API's para completar los diferentes flujos prioritarios.

Tenemos que tener en cuenta también los diferentes datos y parámetros a usar requeridos para realizar las diferentes funciones, como en el caso de un login puede ser el nombre de usuario y la contraseña.

En este ejemplo vamos a tomar como flujos para realizar las pruebas de rendimiento el registro y el login de un usuario.

En nuestro caso además de llamar a las diferentes API's para registro y login, antes de poder hacer un login en nuestra aplicación deberemos de llamar a otro servicio para que active la cuenta del jugador. Es por eso por lo que es prioritario definir previamente los flujos y las llamadas para completar la funcionalidad correctamente y no dejar escapar ningún paso.

Una vez tenemos definidos estos flujos principales para realizar los tests de rendimiento, tenemos que tener claros cuales son los criterios de aceptación. Con estos criterios de aceptación ya podremos tener una base para decidir cuanto de estable es nuestra aplicación y comprobar en que hay que mejorar.

Ahora vamos a ver como implementar dichos flujos en JMeter. A modo de ejemplo para guiar al lector vamos a realizar las pruebas de rendimiento sobre el registro y posterior login en la aplicación.

4.5.2 Creación de un plan de pruebas

Lo primero que nos encontramos al abrir el frontal de la herramienta es un plan de pruebas vacío. Podemos empezar dicho plan de pruebas desde 0 si sabemos lo que vamos a ir necesitando o cargando una plantilla que nos sirva de guía.

En nuestro caso vamos a cargar una plantilla que nos sirva de guía y la vamos a ir modificando. Para ello hacemos clic en Files → Templates y escogemos por ejemplo "Building a Web Test Plan" y le daremos al botón create.

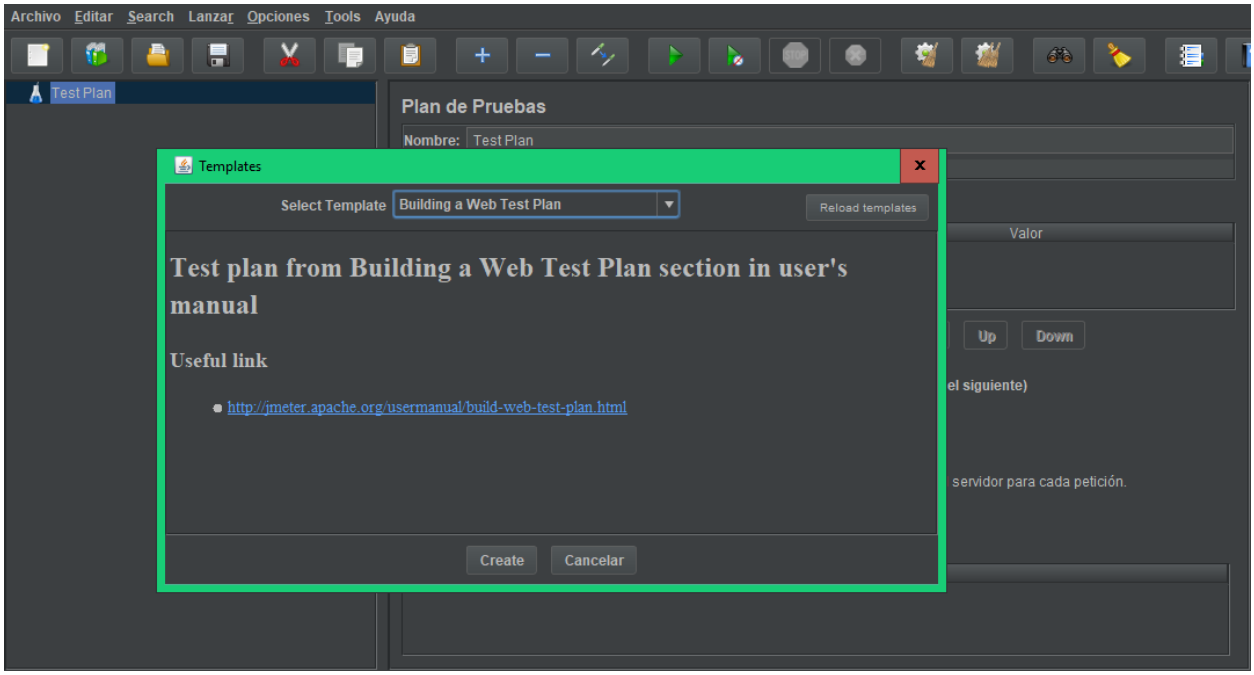


Ilustración 36 Template

Una vez creada dicha plantilla vamos a ir viendo los diferentes componentes creados y vamos a ir actualizando dicha plantilla en función de nuestras necesidades.

4.5.2.1 Grupo de hilos

Lo primero que vemos al crear la plantilla es un Grupo de Hilos, en el que cada hilo es un usuario virtual. Aquí podemos ver que estos usuarios son un parámetro configurable bastante importante a la hora de hacer nuestros test de rendimiento, al igual que el Periodo de Subida que nos servirá para poner un intervalo de tiempo para ir incrementando los hilos que configuremos y otros tantos parámetros en relación a tiempos y repeticiones de las peticiones que configuremos.

En este ejemplo de pruebas de rendimiento nos vamos a ir basando en estos dos parámetros para ver como reacciona la aplicación ante diferente carga de usuarios y podremos ver con que carga la aplicación se comportaría correctamente y a partir de con cuantos usuarios ya empieza a tambalear la estabilidad de la aplicación.

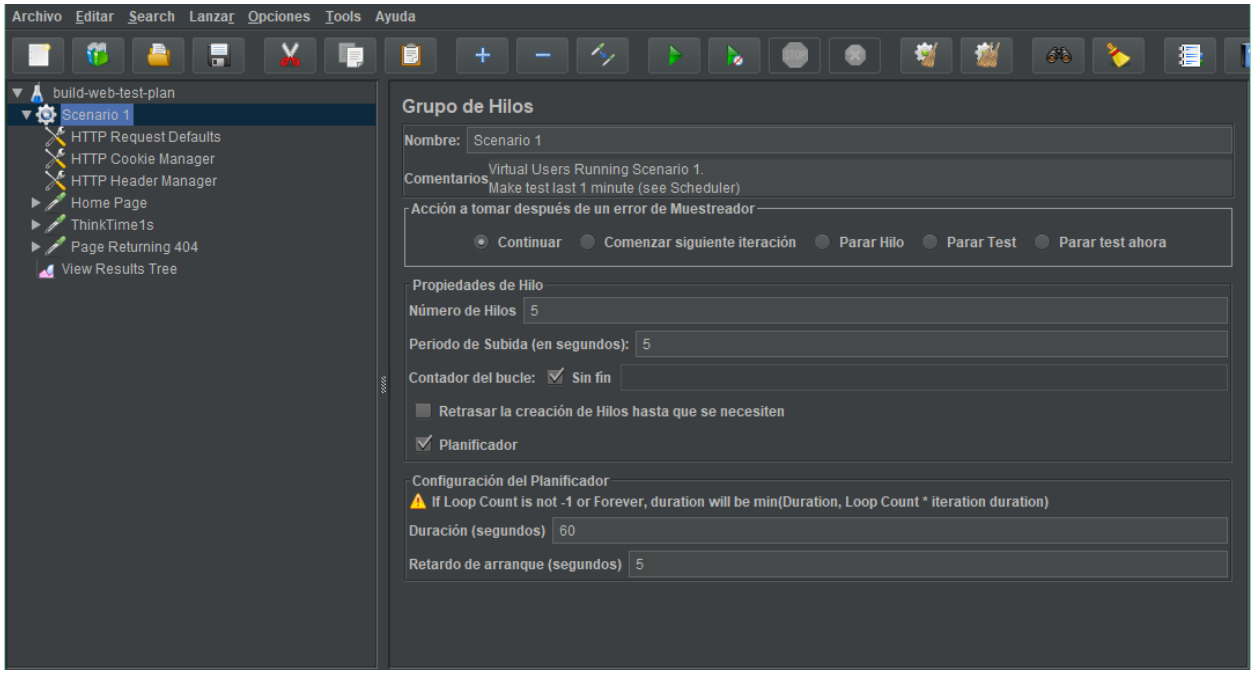


Ilustración 37 Template cargada

En mi caso, el siguiente campo HTTP Request Default no nos va a hacer falta con lo cual lo puedo eliminar y no entraremos en más detalles.

4.5.2.2 Configuración de variables

Lo que si me va a hacer falta para completar la funcionalidad del registro son dos variables configuradas de antemano. Estas son los usernames de los usuarios nuevos de la plataforma. Se formarán con una variable que va a tener un valor constante “usuarioPruebas” y otra variable random que será un número del 100 al 15.000 que acompañará a nuestra constante para así crear usuarios diferentes ya que no puede haber usuarios con el mismo username en nuestra plataforma.

Para ello vamos a hacer clic con el botón derecho en el Grupo de Hilos y vamos a seleccionar Añadir → Elementos de Configuración → Variable definida por el Usuario y Variable aleatoria. Las configuraríamos de la siguiente forma:

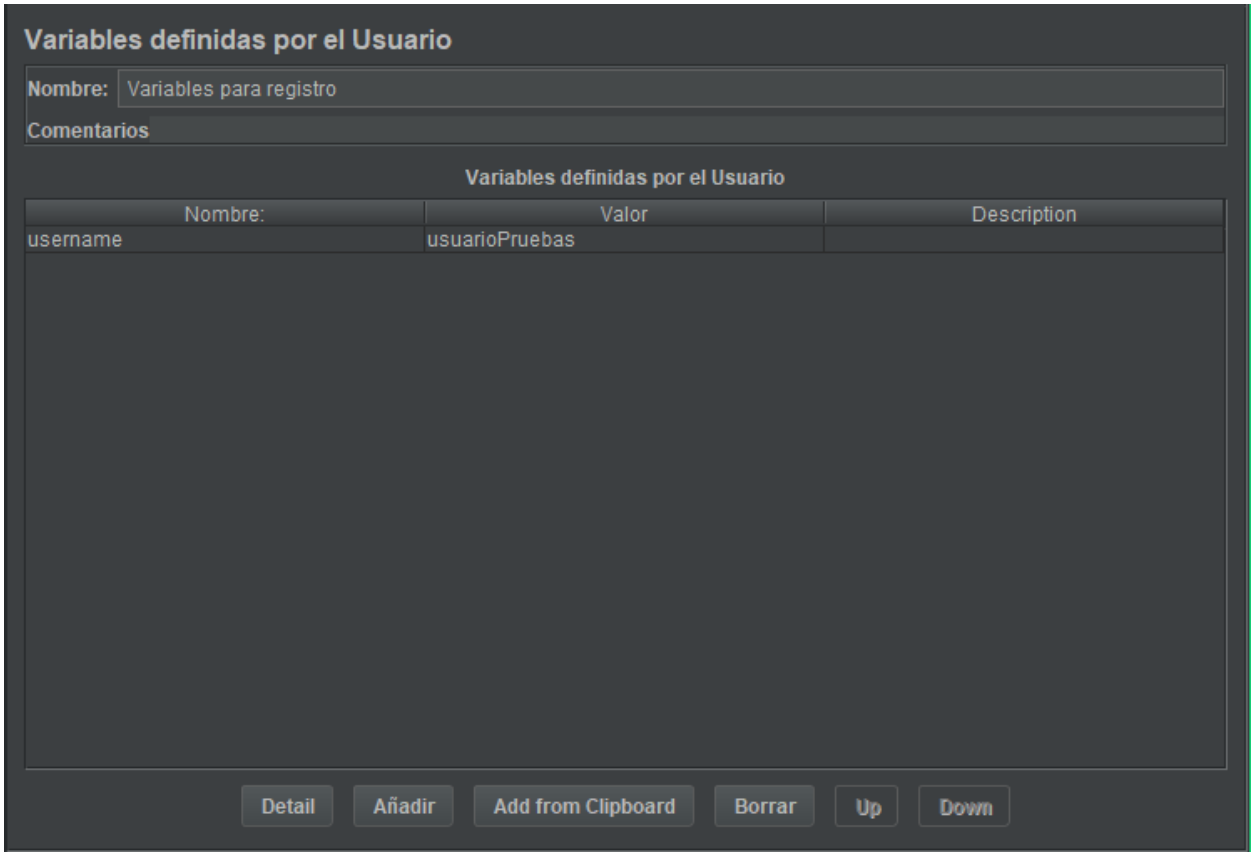


Ilustración 38 Variable constante username

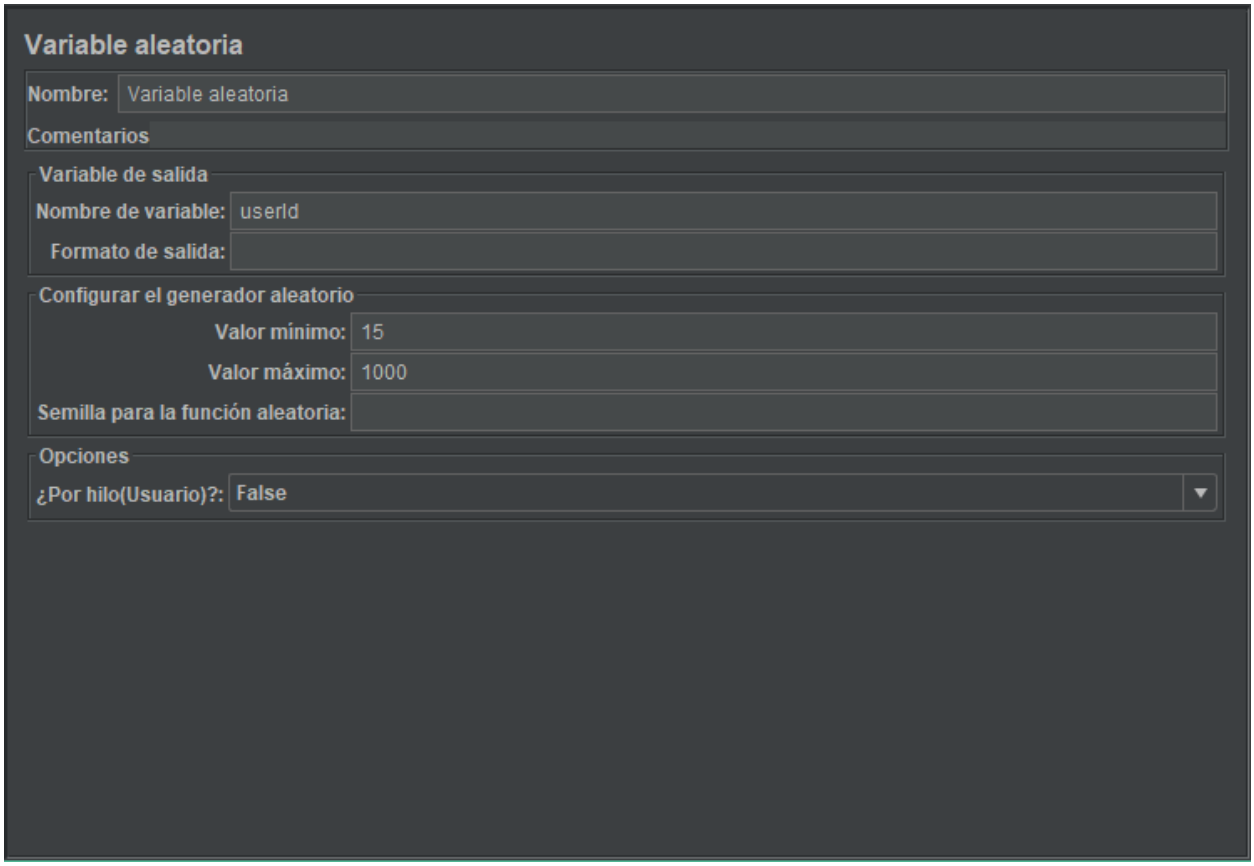


Ilustración 39 Variable aleatoria username

Una vez configuradas las variables que nos hacen falta para montar estos tests pasaremos al siguiente apartado, el Gestor de cookies.

4.5.2.3 Gestor de cookies

Como nuestra aplicación utiliza cookies deberemos de dejar el Gestor de Cookies en nuestro proyecto. También se puede añadir de forma manual mediante Add → Config Element → HTTP Cookie Manager.

Una vez tenemos estas tres secciones completas vamos a pasar al siguiente componente que nos encontramos, el Gestor de Cabecera HTTP.

4.5.2.4 Gestor de cabecera HTTP

Este Gestor de Cabecera HTTP tiene la particularidad de que tendremos que añadir un parámetro de autorización, llamado token, del que muchas webs hacen uso para la identificación de los usuarios y asegurar la seguridad de las cuentas de los mismos, ya que son temas muy delicados.

Se le otorga un token a cada usuario mediante el cual debe de identificarse mandándolo en la cabecera HTTP. Para sucesivas llamadas después de hacer login se guardará dicho token y se irá enviando mediante esta cabecera. Este token se extraerá de la llamada a la API de Login de la que más tarde hablaremos. La configuración de este gestor de cabecera será la siguiente:

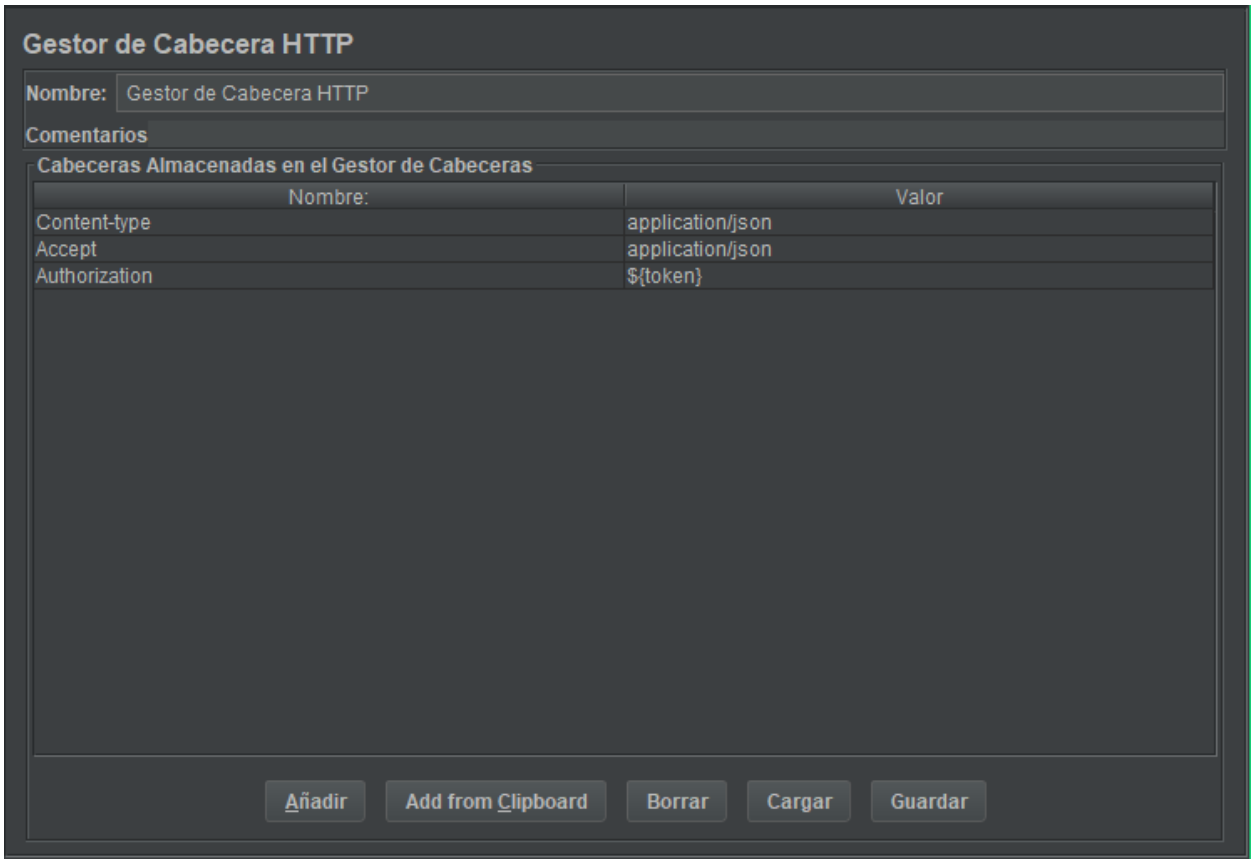


Ilustración 40 Gestor cabeceras HTTP

4.5.2.5 Llamadas HTTP

Después de configurar estos parámetros vamos a implementar las llamadas HTTP para completar los diferentes servicios. En nuestro caso necesitaríamos la llamada al servicio de registro, la de activación de la cuenta del jugador, la del login de usuario y antes que todo eso tendremos que generar un dni aleatorio, ya que como sucede con los usernames, no puede haber más de un jugador con un mismo dni. Es por eso por lo que llamaremos a un servicio externo que nos generará un dni aleatorio para usarlo en el registro de usuarios.

Las llamadas HTTP se añaden haciendo clic con el botón derecho sobre el Grupo de Hilos y seleccionando Añadir → Muestreador → Llamada HTTP.

Una vez añadidas estas llamadas nos quedaría así la estructura del proyecto, aunque más tarde modificaremos la sección de resultados para ver mejor los resultados arrojados de estas llamadas.

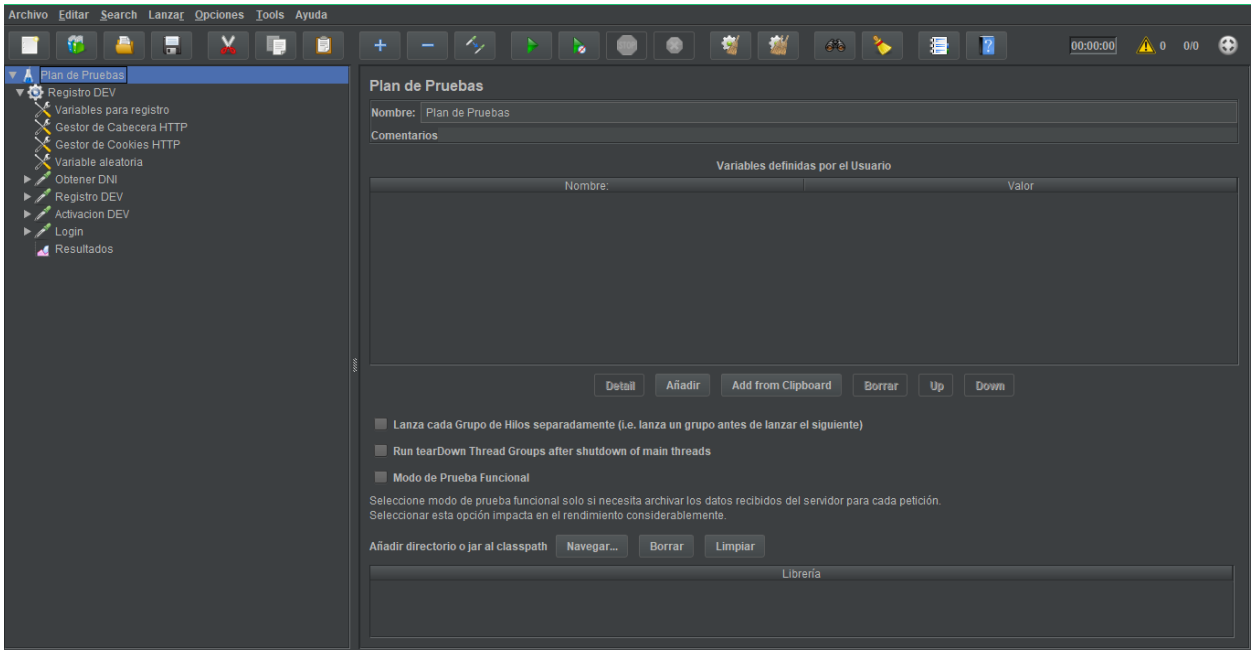


Ilustración 41 Registro y login de un usuario

4.5.2.5.1 Obtener DNI

El siguiente paso será ir revisando las diferentes llamadas creadas e ir completándolas. Vamos a empezar por la llamada de Obtener DNI. Esta llamada se implementará con el objetivo de obtener un dni aleatorio y válido para poder ir registrando usuarios en la plataforma sin duplicar los documentos de identidad, ya que es una validación obligatoria en la plataforma. Para ello se va a llamar a una web externa llamada http://www.generador-de-dni.com.

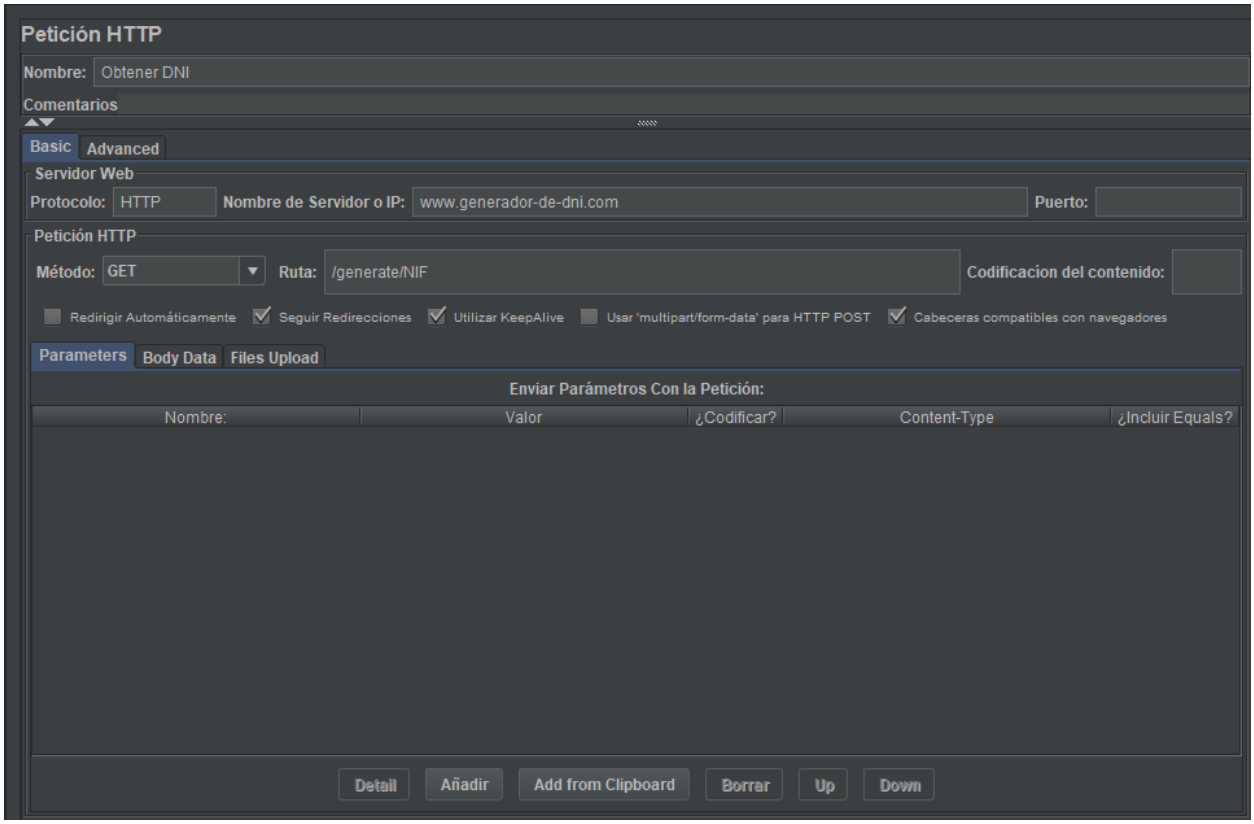


Ilustración 42 Petición HTTP obtener DNI

Se hará una petición GET para obtener el dato requerido, en el que el protocolo usado será HTTP y añadiremos la ruta de la que podemos consumir ese servicio que será /generate/NIF.

Además, tendremos que añadir un extractor JSON para extraer el DNI devuelto en la respuesta a esta llamada HTTP para así poder reutilizarlo en las siguientes llamadas. Se añadirá haciendo clic con el botón derecho encima de la petición HTTP y seleccionando Añadir → Post Procesadores → JSON Extractor.

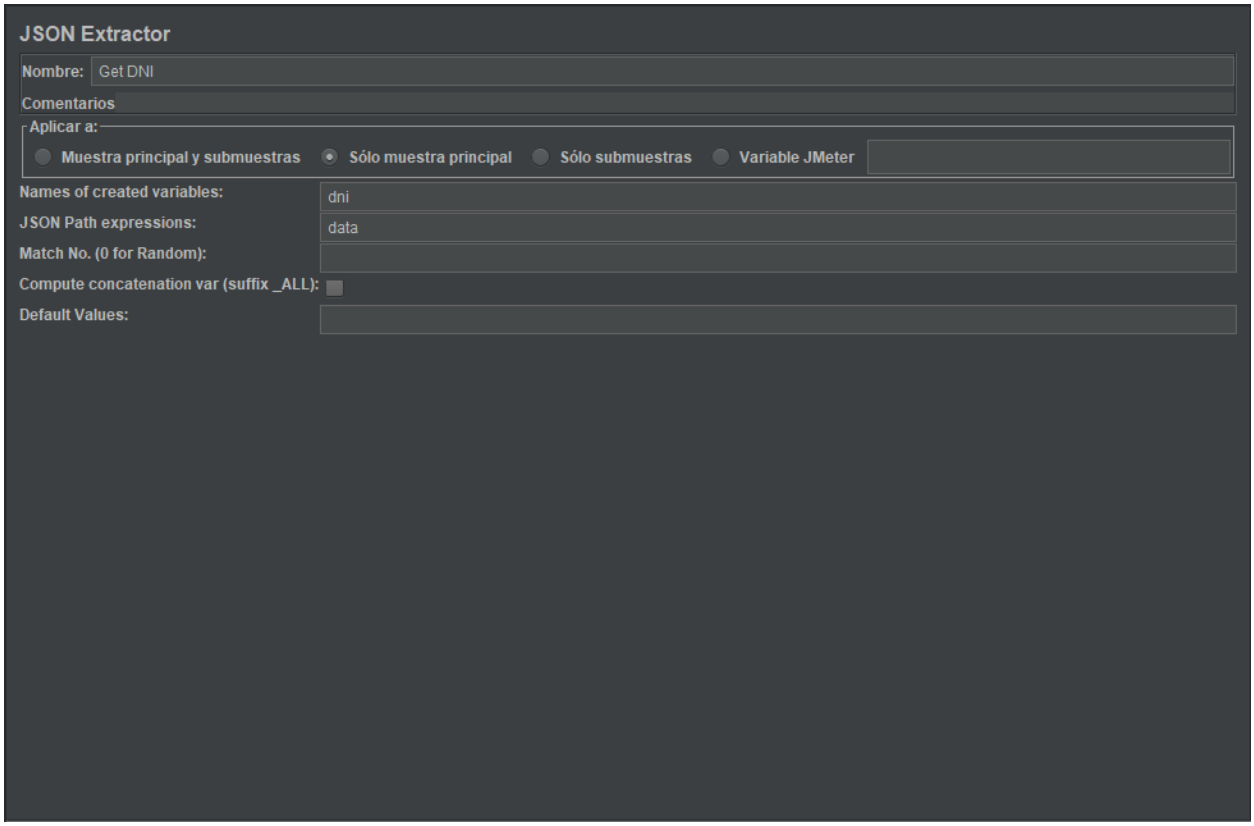


Ilustración 43 Extracción DNI

Con esto ya tendríamos el primer paso para el registro de nuestro jugador. Ahora vamos a ir a completar la siguiente llamada de nuestro flujo que sería la correspondiente al registro.

4.5.2.5.2 Llamada registro

Se hará una petición POST para registrar a un usuario con el DNI obtenido de la anterior petición. El protocolo usado será HTTP y añadiremos la ruta como en la anterior llamada. Esta llamada tiene de particular que necesita enviar unos parámetros para el registro del usuario como pueden ser el nombre, mail, DNI,... En resumen datos personales del usuario para crearse una cuenta. Estos datos irán en el Body. La mayoría de ellos serán constantes y datos únicos como pueden ser mail, usuario y DNI, serán datos variables creados a partir de llamadas posteriores o las variables definidas anteriormente.

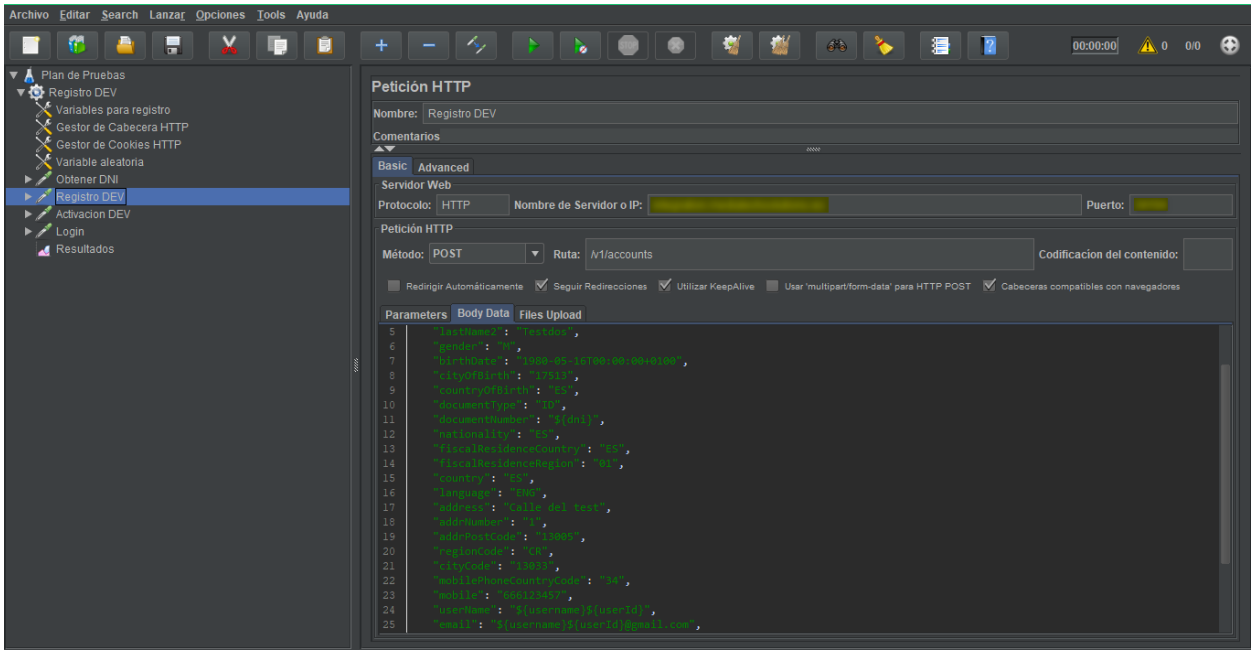


Ilustración 44 Llamada al servicio de registro

Como la anterior llamada, de esta llamada también sacaremos un parámetro que es el código de la cuenta. Este parámetro lo reutilizaremos para la activación de la cuenta. Se hará de la misma forma que con el DNI, con un JSON Extractor.

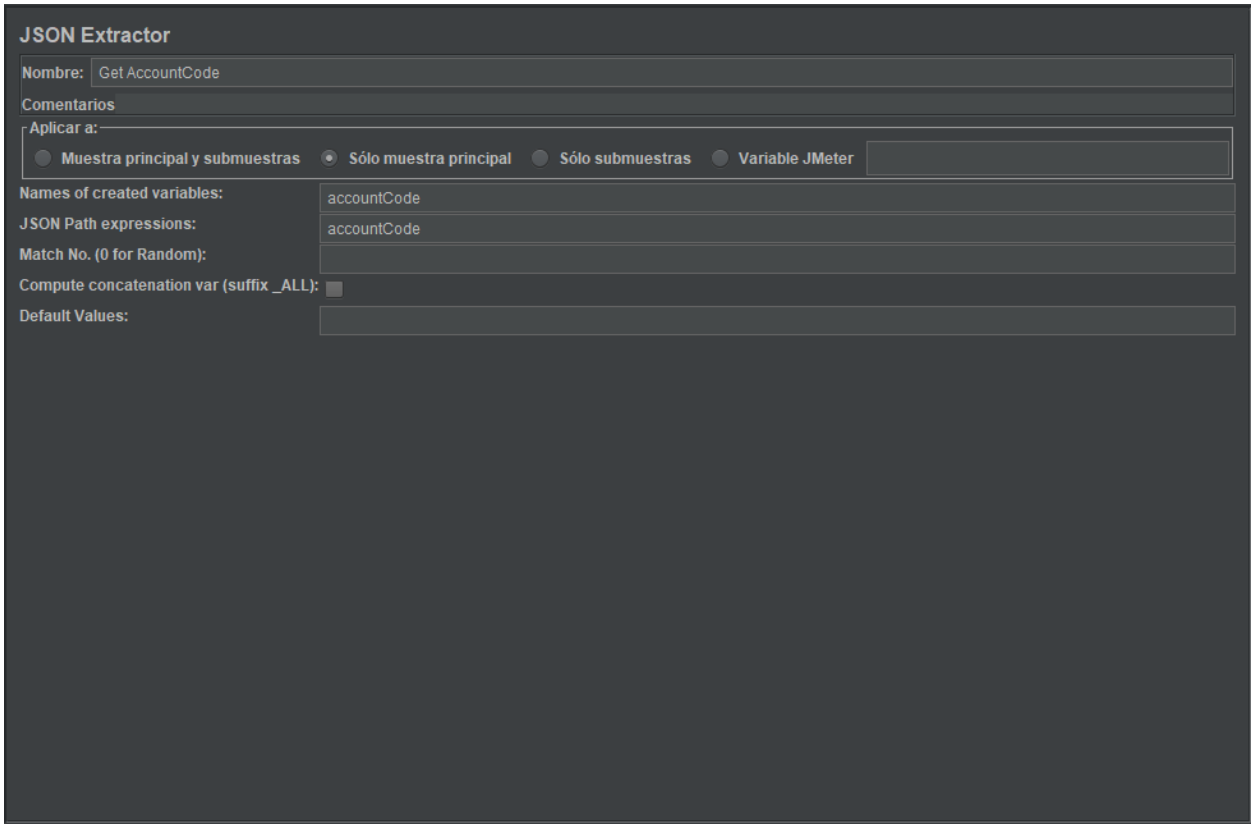


Ilustración 45 Extracción del código de la cuenta

4.5.2.5.3 Llamada activación

El siguiente paso será activar la cuenta. Para ello implementaremos otra llamada de tipo POST, en la que utilizaremos el código de la cuenta de la llamada anterior y lo enviaremos en el Body.

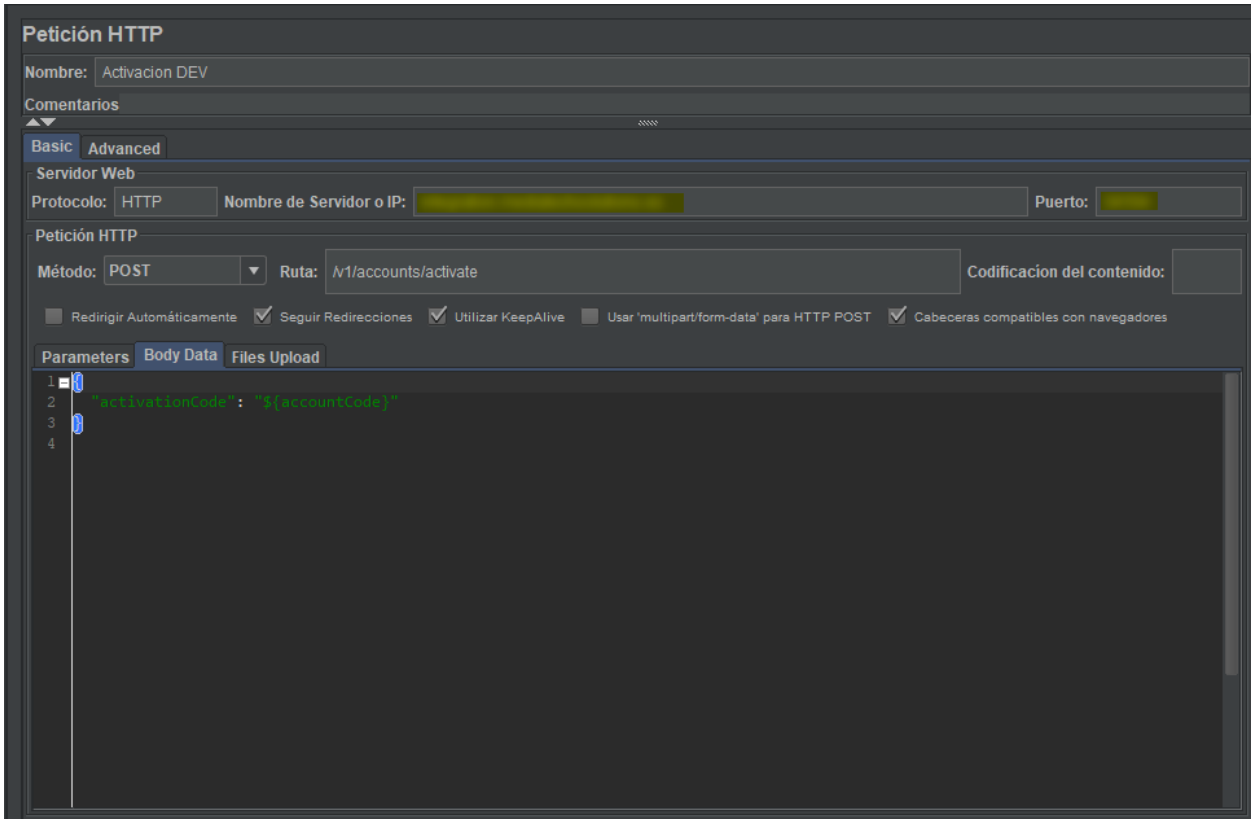


Ilustración 46 Llamada para la activación del usuario

Una vez activamos la cuenta, el último paso es implementar la funcionalidad del Login para completar el flujo y comprobar que el usuario se ha registrado, está activo y puede realizar las diferentes funciones que la plataforma nos ofrezca.

4.5.2.5.4 Llamada login

Es una llamada como las anteriores y en este caso el Body contiene los datos identificativos del usuario para hacer un login. Además, de esta llamada se extrae el token que autentifica al usuario ante cualquier movimiento por parte de este.

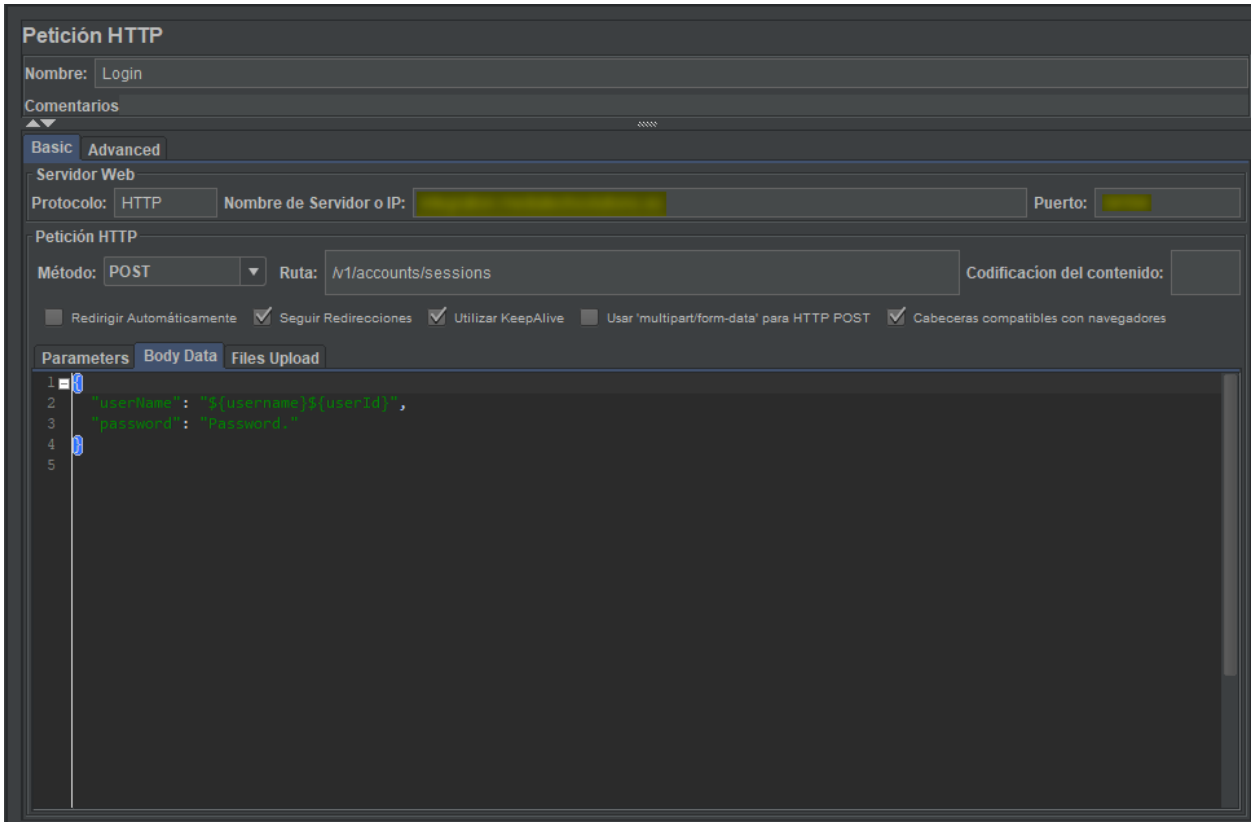


Ilustración 47 Llamada para realizar un login

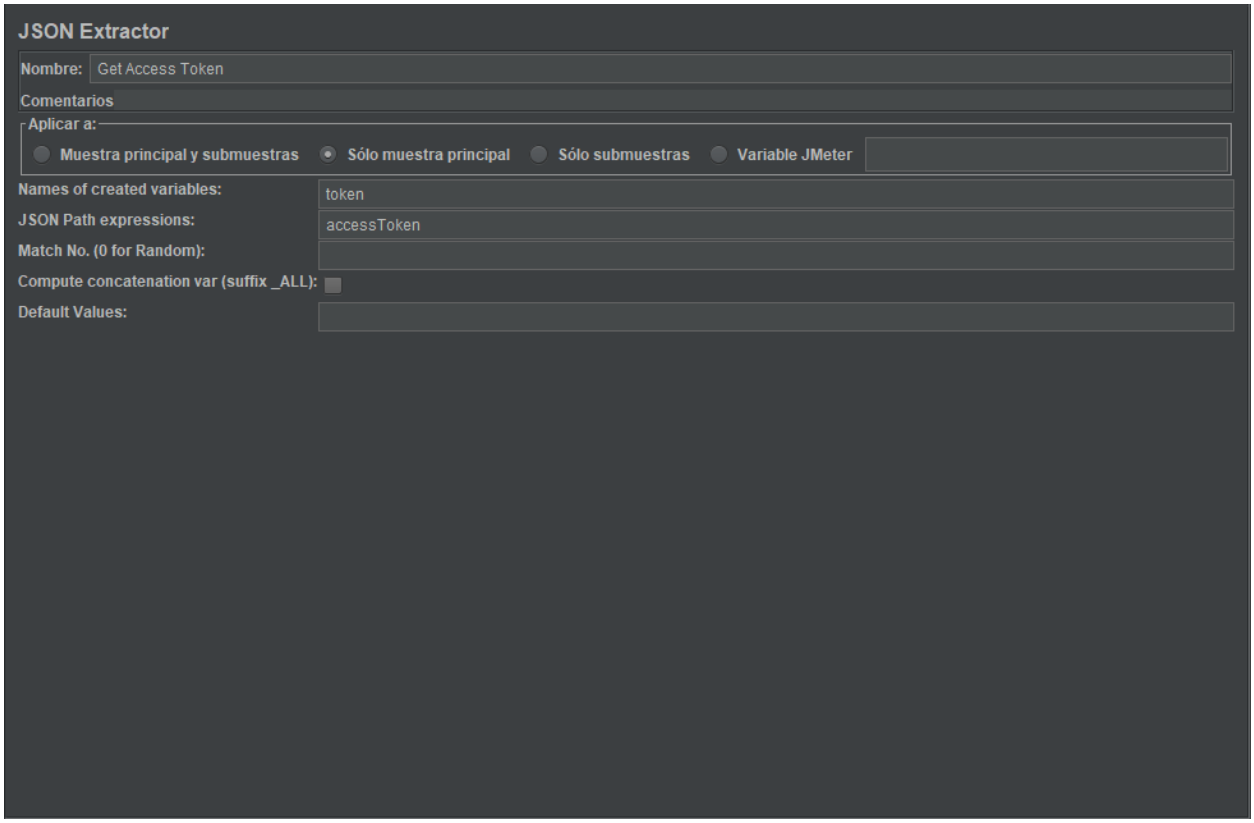


Ilustración 48 Extracción del token

Una vez implementadas todas las llamadas, lo único que nos queda es adjuntar algún elemento para visualizar los resultados de lanzar las pruebas.

4.5.2.5.5 Análisis de los resultados

En el template viene por defecto un árbol de resultados en el que se verán las diferentes llamadas con su respuesta asociada. Además, vamos a añadir un elemento que nos hará un resumen de todas las llamadas que han ido bien y las que han ido mal en porcentajes. Para esto, seleccionamos Añadir → Receptor → Reporte resumen.

Ya que tenemos todo montado, lo que vamos a hacer es definir dos casos de pruebas para ejecutarlos sobre la plataforma. Estos casos de prueba van a variar en cuanto al número de usuarios, aumentándolos para ver como se comporta la aplicación.

Empezaremos añadiendo 10 hilos, el cual es un escenario realista ya que puede haber fáclmente 10 personas registrándose a la vez en una plataforma de este calibre, y seguiremos con un escenario no tan realista como son 300 hilos, el cual es muy poco probable que se dé a la vez.

Resultado de 10 hilos:

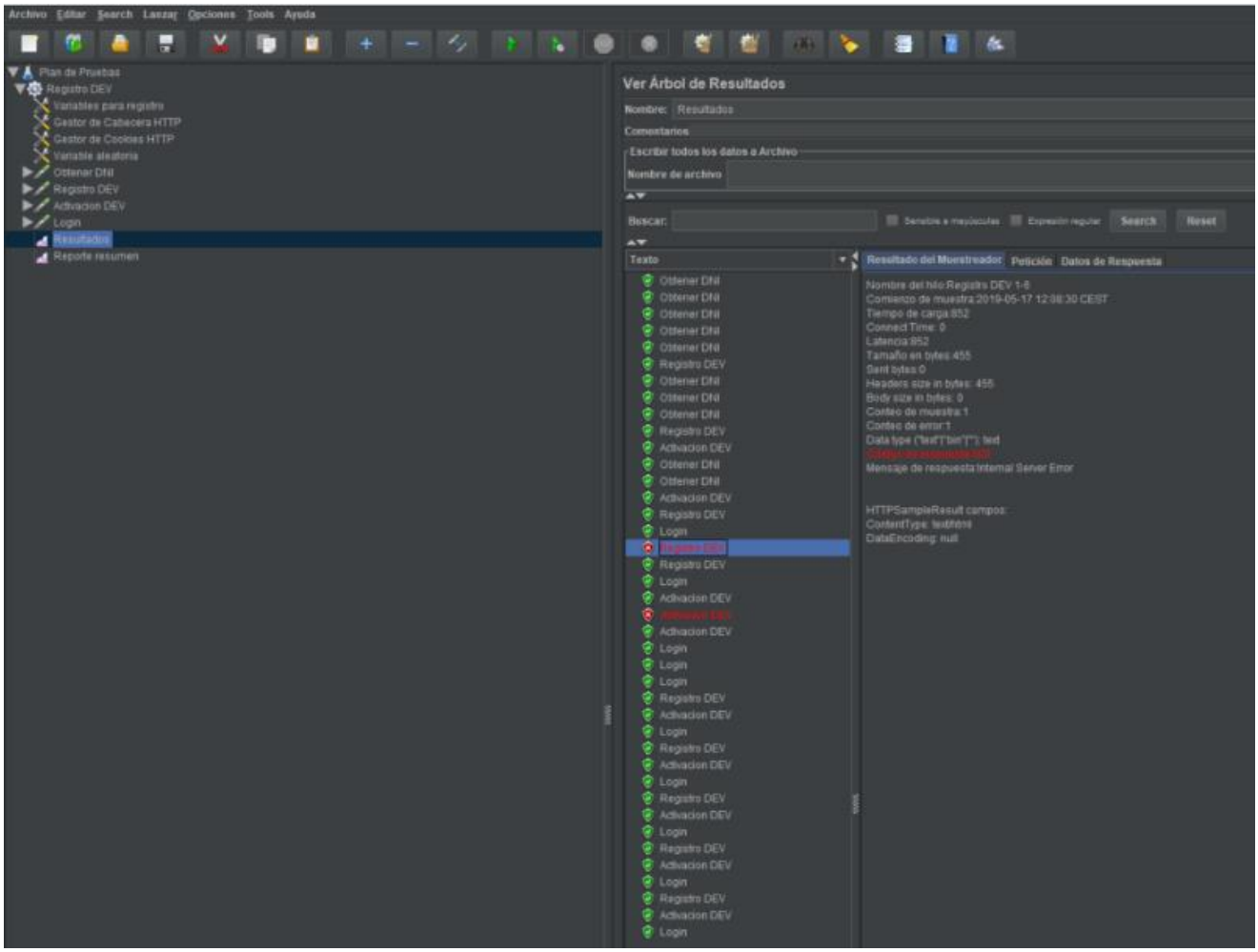
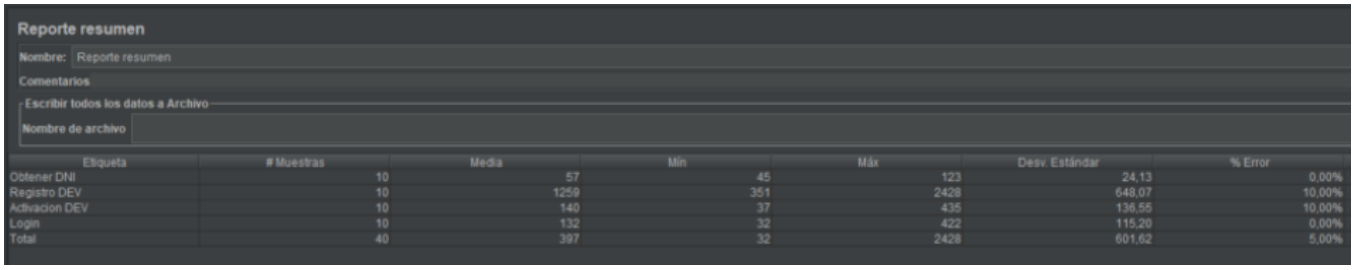


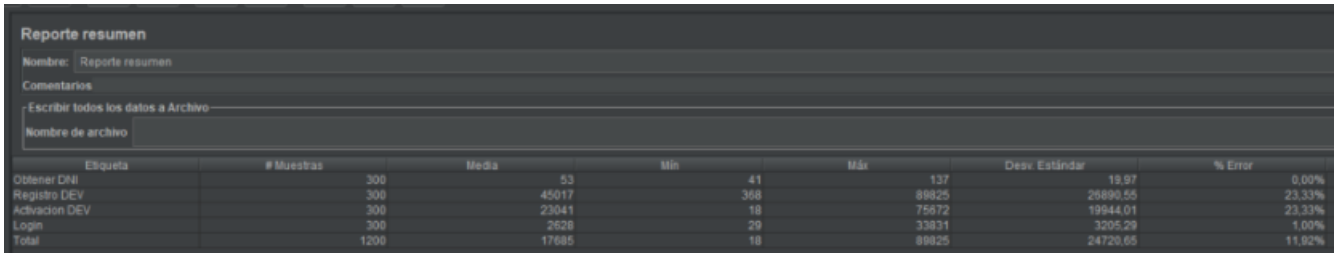
Ilustración 49 Resultado peticiones 10 hilos



Etiqueta	# Muestras	Media	Mín	Máx	Desv. Estándar	% Error
Obtener DNI	10	57	45	123	24,13	0,00%
Registro DEV	10	1259	351	2428	648,07	10,00%
Activacion DEV	10	140	37	435	136,55	10,00%
Login	10	132	32	422	115,20	0,00%
Total	40	397	32	2428	601,62	5,00%

Ilustración 50 Resumen peticiones 10 hilos

Resultado 300 hilos:



Reporte resumen

Nombre: Reporte resumen

Comentarios

Escribir todos los datos a Archivo

Nombre de archivo

Etiqueta	# Muestras	Media	Mín	Máx	Desv. Estándar	% Error
Obtener DNI	300	53	41	137	19,97	0,00%
Registro DEV	300	45017	368	89825	26899,55	23,33%
Activacion DEV	300	23041	18	75672	19944,01	23,33%
Login	300	2628	29	33831	3205,29	1,00%
Total	1200	17685	18	89825	24720,65	11,92%

Ilustración 51 Resumen peticiones 300 hilos

Como podemos ver con 10 hilos sólo ha fallado una de las peticiones lo cual nos supone como se ve en la imagen un 10% de error medio. Son valores aceptables teniendo en cuenta que no es un entorno de producción y que no dispone de tantos recursos como este.

Cuando metemos los 300 hilos ya nos damos cuenta que la tasa de error se maximiza hasta llegar a más del doble que la anterior. Este resultado es más preocupante y habría que investigar el porqué de esa tasa de errores para darle una solución a corto o medio plazo ya que como antes dijimos no es una situación muy realista que 300 usuarios se registren a la vez en nuestra plataforma.

5 PRUEBAS FUNCIONALES

5.1 Definición de prueba funcional

Como su propio nombre indica, las pruebas funcionales son aquellas que tienen como objetivo verificar las distintas funcionalidades que integra una aplicación software. Esta verificación se hará entorno a unos requisitos de aceptación, los cuales se propondrán antes de diseñar una nueva funcionalidad. Aquí sólo se tienen en cuenta las entradas al sistema y las salidas que este proporciona a esas entradas dadas, no se tiene en cuenta código fuente ni nada por el estilo, es lo que antes hemos nombrado como pruebas de caja negra.

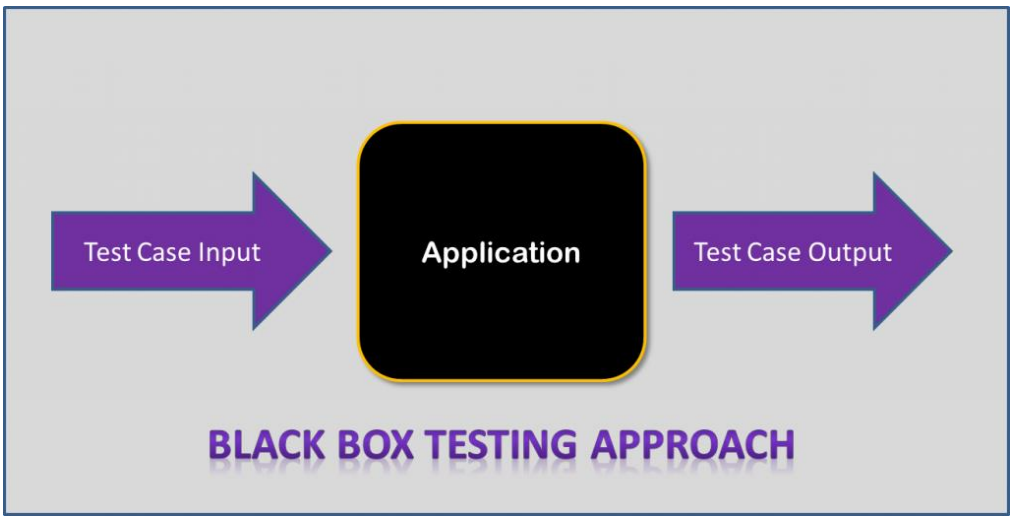


Ilustración 52 Pruebas de caja negra

https://www.panel.es/blog/software-qa-cuales-son-los-tipos-de-pruebas-software/

Estas pruebas suelen implicar tanto comprobaciones en la interfaz de usuario de la aplicación, como API's, pruebas con bases de datos, aplicaciones cliente/servidor…

Este tipo de pruebas se pueden hacer de dos formas, manuales y automáticas con diferentes frameworks y lenguajes, los cuales veremos en apartados siguientes, dependiendo de la naturaleza de la aplicación y nuestras necesidades.

En este tipo de pruebas además de unas entradas conocidas también podemos tener unas precondiciones que se deben ejecutar antes de iniciarse la prueba.

5.2 Principales diferencias entre pruebas funcionales y no funcionales

En este punto vamos a abordar las principales diferencias entre las pruebas funcionales que estamos tratando en este capítulo y las pruebas no funcionales. Los test no funcionales son pruebas que verifican aspectos no funcionales de nuestra aplicación, en capítulos anteriores hemos visto las pruebas de rendimiento y las de seguridad, estas dos son un gran ejemplo de lo que son este tipo de pruebas.

A continuación vamos a hacer un pequeño resumen de las diferencias más significativas entre estos dos grandes grupos:

1. En cuanto a la ejecución de estos tipos de pruebas, las funcionales suelen ser ejecutadas en primer lugar. Una vez comprobada la funcionalidad de la aplicación ya se pasarían a controlar los demás parámetros.

2. Las pruebas funcionales se basan en requisitos que el propio cliente pone, mientras que las no funcionales se basan en unas expectativas.

3. Las funcionales se ejecutan para dar el visto bueno a acciones del software mientras que las otras se ejecutan para validar el rendimiento de este.

4. Las pruebas funcionales se pueden ejecutar tanto manualmente como automáticamente como antes mencionamos, mientras que las otras casi siempre se suelen ejecutar de forma automática debido a su complejidad y a las facilidades que nos proporcionan las diferentes herramientas de automatización.

5. Las primeras en resumen describen lo que hace el software y las otras cómo funciona.

A continuación vamos a ver un listado, en el que se catalogan distintos tipos de pruebas según sean funcionales o no funcionales:

Pruebas funcionales	Pruebas no funcionales
• Pruebas unitarias • Pruebas de humo (También conocidos como smoke tests, se componen de un conjunto no exhaustivo de tests cuyo objetivo es garantizar el correcto funcionamiento de las funcionalidades más importantes de la aplicación) • Pruebas de aceptación	• Pruebas de carga • Pruebas de estrés • Pruebas de volumen (Consiste en comprobar el funcionamiento normal de la aplicación con ciertos volúmenes de datos dados) • Pruebas de configuración (Sirven para comprobar que efectos traen consigo un

• Pruebas de integración • Pruebas de regresión • Pruebas de sanidad (También conocidas como sanity tests, es un conjunto de pruebas más exhaustivas para asegurar que parte del sistema funcione)	cambio en la configuración de la aplicación) • Pruebas de usabilidad (Para evaluar que tan fácil de usar es una determinada aplicación) • Pruebas de seguridad • Pruebas de resistencia • Pruebas de recuperación (Para verificar el tiempo de respuesta en caso de que una aplicación sufra un fallo tanto hardware como software) • Pruebas de mantenibilidad (Pruebas para comprobar que tan fácil es mantener una aplicación)

Tabla 2 Tabla clasificación pruebas funcionales y no funcionales

5.3 Fases en la ejecución de pruebas funcionales

La ejecución de este tipo de pruebas suele seguir un procedimiento bastante marcado centrado principalmente en 5 puntos en los cuales intervienen varias personas además del propio tester, como puede ser el cliente, el Scrum Master, el Bussiness Analyst…

- **Scrum Master:** Actúa como un facilitador, es el encargado de llevar los bloqueos y promueve acciones y reuniones que invitan a pensar y reflexionar sobre posibles mejoras, trabajo realizado, planificación de trabajo [13]…
- **Bussiness Analyst:** Esta persona es la que posee los conocimientos técnicos sobre la aplicación software. Su principal objetivo es la de ser el interlocutor entre el usuario y el departamento de sistemas.

Una vez definidos estos roles, vamos a ver las fases en la ejecución de una prueba funcional [14]:

1. **Ánalisis de requerimientos:** En esta fase se hace un estudio de la documentación tanto entregada por el cliente como por el jefe de proyecto. Con este estudio se irá recopilando información la cual nos servirá para conocer la funcionalidad a validar, hacer un planning de lo que habrá que probar, ver las distintas casuísticas…

2. **Elaboración del plan de pruebas:** En esta elaboración se va a generar un documento en el que se detallarán las consideraciones para la validación de esta funcionalidad (riesgos, datos, entornos necesarios…).

3. **Elaboración de los casos de pruebas:** Aquí se diseñarán los diferentes test que tienen que ser ejecutados contra la aplicación para comprobar la correcta funcionalidad en sus múltiples caminos y combinaciones teniendo en cuenta los requisitos expuestos por el cliente.

4. **Ejecución de los tests:** Esta fase será la de ejecutar los casos de prueba descritos en la fase anterior y recoger los resultados que arroja la aplicación software. Además de esto se podrán detectar incidencias en la aplicación mientras se sigue el flujo de ejecución.

5. **Elaboración de reportes:** En este paso final se generará un reporte con los resultados finales y los resultados esperados haciendo una comparativa y viendo los test ejecutados con éxito y los fallidos.

5.4 Herramientas utilizadas

En este penúltimo apartado del capítulo vamos a seguir en la misma línea y nos vamos a centrar en nombrar diferentes herramientas que nos ayudarán con estas tareas de automatización de las pruebas funcionales. Se verán tanto frameworks diseñados para las pruebas de aplicaciones móviles como herramientas para el desarrollo de las mismas en aplicaciones web.

Como dijimos en anteriores capítulos, estas pruebas también se podrían hacer de forma manual pero existen muchas ventajas que nos da esta automatización. Un ejemplo claro de esta sería para efectuar las regresiones, que no es más que comprobar las funcionalidades principales de la aplicación teniendo en cuenta el impacto de los nuevos desarrollos llevados a cabo y los defectos solucionados. De esta forma nos garantizamos probar estas funcionalidades y detectar fallos que podrían ser catastróficos en nuestra aplicación. Con lo cual ahorramos tiempo y dinero que son dos de los pilares fundamentales a la hora de llevar a cabo un proyecto.

Vamos a ir viendo alguna de las herramientas más populares para este fin:

5.4.1 Selenium

Esta es la herramienta de automatización de pruebas funcionales en aplicaciones web más conocida y más potente en la actualidad.

Fue creado en un principio por Jason Huggins en el año 2004. Aunque fue creado por él, pronto se unieron a la causa más personas especializadas en pruebas y en desarrollo software.

Es muy importante destacar que además de ser la herramienta más potente en cuanto a automatización, es un software gratuito (Open Source).

Esta herramienta además ofrece muchas facilidades para gente novata o inexperta en el tema del desarrollo y lenguajes de programación. Además de poder escribir nuestras pruebas en múltiples lenguajes como pueden ser Java, C#, Ruby, Groovy, Perl, Php o Python, también nos provee de una herramienta mediante la cual se pueden grabar las acciones que vaya realizando el usuario contra la aplicación web y mediante estas acciones se creará la prueba.

También es un punto a destacar el que se pueden ejecutar dichas pruebas en la mayoría de los navegadores que hoy en día manejamos, como Firefox, Chrome, IE…Además también es compatible con múltiples Sistemas Operativos.

Ilustración 53 Selenium WebDriver

https://www.seleniumhq.org/projects/webdriver/

Este framework se compone de diferentes componentes los cuales se van a exponer a continuación:

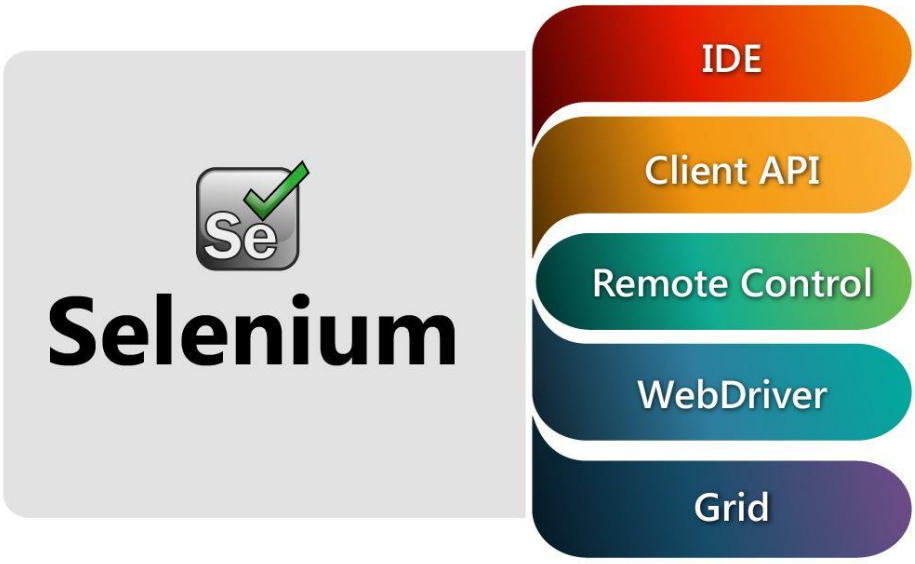


Ilustración 54 Componentes Selenium

https://www.kovair.com/blog/selenium-3-0-boost-software-testing-automation/

1. **Selenium IDE**

Es una herramienta que nos va a ayudar en la ejecución de test sobre una página o aplicación web.

Nos permite desarrollar nuestros propios test para la comprobación de funcionalidades en nuestra aplicación. Como hemos mencionado antes, esta es una herramienta que facilita el desarrollo de pruebas automáticas para alguien con poca experiencia en el mundo de la automatización ya que proporciona una gran variedad de comandos o funciones y además permite grabar acciones que se realicen contra la aplicación.

Es un entorno de desarrollo integrado en Selenium. Lo que hace es que permite grabar, editar y depurar pruebas. Con lo cual mediante acciones sobre la aplicación podríamos crear el script de la prueba sin tener conocimientos sobre lenguajes de programación ni nada por el estilo, aunque es bastante recomendable el tener conocimientos sobre estos ya que pueden surgir pequeños errores y para solucionarlos y depurarlos conviene tener un mínimo de formación en esta área.

Este componente está implementado como una extensión de Firefox y los script se generan en un lenguaje especial para Selenium, Selenease. Este lenguaje provee de comandos para la realización de múltiples acciones, desde hacer clic en un elemento de la aplicación, hasta comprobaciones de texto, scroll…

Se incluye en este componente una herramienta para la depuración de los test que desarrollemos.

Las características principales de este componente son:

- Grabación y reproducción fácil.
- Selección inteligente de campos usando ID, nombre o XPath según se necesite. Es más conveniente el uso de ID para identificar un elemento ya que es un identificador único, mientras que la identificación por XPath nos puede conducir a errores ya que se basa en la posición que ocupa el elemento en un determinado tiempo.
- Autocompletado de los comandos de Selenium más comunes.
- Pruebas de revisión cruzada.
- Depuración y puntos de verificación (breakpoint).
- Almacenar las pruebas como Selanese, Ruby, Java y otros formatos.
- Soporte al archivo user-extensions.js.
- Opción para asertar el título de la página.
- Opción de modificarle a la medida con el uso de complementos.

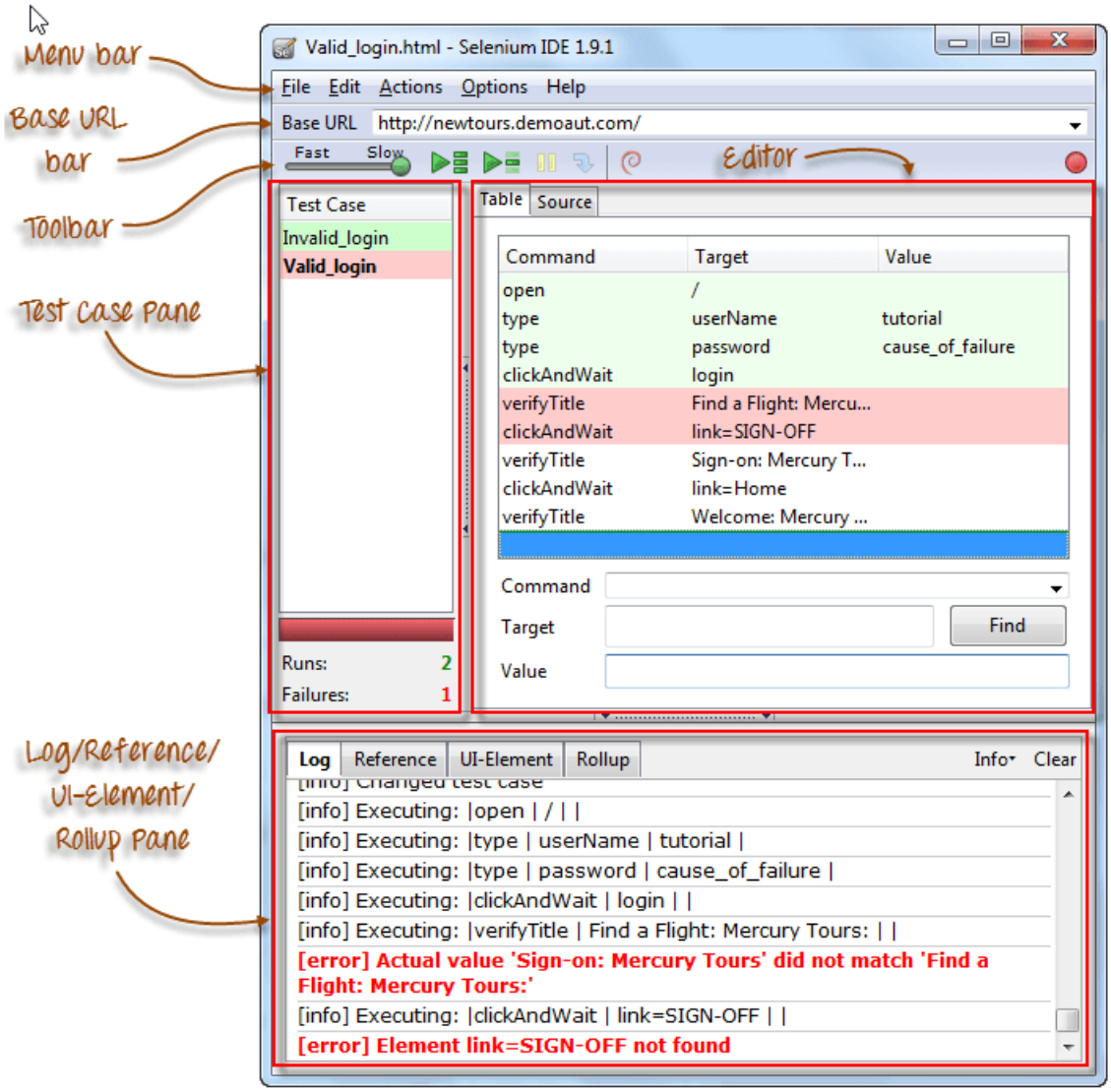


Ilustración 55 Interfaz Selenium IDE

https://www.guru99.com/introduction-selenuim-ide.html

Como podemos ver en la imagen anterior, la interfaz de esta herramienta es bastante sencilla. Se pueden distinguir 6 partes principales dentro de ella:

- Barra de menú, como en casi toda herramienta informática con sus posibles acciones a realizar y configuraciones.

- URL de la página, dirección de la página principal sobre la que se está ejecutando el test.

- Barra de acceso rápido, la cual la integran un conjunto de botones para realizar acciones de forma rápida y sencilla como sería la ejecución de un test.

- Sección de casos de prueba, en la que se muestran todos los test que hemos desarrollado y más abajo se pueden visualizar las ejecuciones de los mismos, viendo cuantos han sido ejecutados correctamente y cuantos han sido fallidos.

- Editor, listado de comandos de Selenium IDE que componen un test definido por el usuario o cargado en la aplicación. Se puede acceder al código del test.

- Log, muestra las ejecuciones de los test paso por paso además de diferente información relacionada con la configuración cuando se lanza un test y nos avisa de los errores que suceden durante una ejecución.

2. Selenium Client API

Es una interfaz de desarrollo de aplicaciones (API) de clientes. Esta sirve como alternativa a escribir pruebas en Selanese.

Provee API para diferentes lenguajes como son Java, C#, Ruby y Phyton.

3. Selenium Remote Control

Es un servidor que acepta comandos para automatizar el navegador a través de HTTP. Cuando se usa con el cliente Selenium API, elimina la necesidad de Selenium IDE.

4. Selenium Grid

Es un servidor que permite usar instancias de navegador ejecutándose en máquinas remotas. Permite ejecutar pruebas en paralelo en múltiples máquinas y manejar diferentes versiones y configuraciones de manera centralizada.

5. Selenium WebDriver

Selenium WebDriver es una herramienta que está diseñada para ejecutar las pruebas automáticas operando en los principales navegadores. Esto último que se ha nombrado es una de las principales ventajas frente a Selenium IDE, ya que el anterior era una extensión de Firefox, mientras que este se puede utilizar con diferentes navegadores web debido a los controladores nativos que incluye esta herramienta como pueden ser IE, Chrome, Opera y por supuesto también Firefox. Además permite realizar pruebas para dispositivos móviles, tanto iPhone como dispositivos Android. Utiliza lenguajes como Python, Ruby, Java y C#.

Gracias a WebDriver ya no necesitamos de un navegador web real para lanzar los test sino que utiliza una aplicación basada en HtmlUnit para simular el navegador.

Tiene una interfaz bastante sencilla e intuitiva con dos clases principales, WebDriver, para el control de los distintos navegadores web, y WebElement, para controlar los elementos que contiene la aplicación.

5.4.2 Appium

Appium es una herramienta de libre distribución usada para la automatización de pruebas en aplicaciones móviles. Permite realizar pruebas nativas, híbridas y de aplicaciones web. Además cabe destacar que permite la ejecución de pruebas sobre dispositivos móviles tanto virtuales, como las imágenes que se pueden montar con Android Studio, como los dispositivos físicos propiamente. Es una herramienta que nos ofrece la posibilidad de realizar pruebas multiplataforma, con la misma API se llevan a cabo pruebas para Sistemas Operativos Android e IOS.

Esta herramienta traduce los comandos de la herramienta vista con anterioridad, Selenium WebDriver, en comandos de UIAutomation, en el caso de que el Sistema Operativo sea IOS o en UIAutomator, en el caso de Android.

Hemos mencionado que traduce los comandos de Selenium, con lo cual admite todos los lenguajes de programación de este, Java, Objective-C, JavaScript con node.js, PHP, Ruby, Python, C #, etc.

Ilustración 56 Appium

https://vikramviknowledgesharing.wordpress.com/2017/01/01/appium-mobile-automation/

5.4.3 Espresso

Es otra herramienta para la automatización de pruebas de aplicaciones móviles como Appium. También es de libre distribución. Como Selenium IDE también tiene la facilidad de grabar acciones contra la aplicación para realizar tus test automáticos sin tener que tocar líneas de código.

Se compone de tres componentes:

- ViewMatchers, permite encontrar vista en la jerarquía de vista actual.
- ViewActions, permite realizar acciones en las vistas.
- ViewAssertions, permite confirmar el estado de una vista.

Viendo lo anterior podemos organizar los pasos para realizar un test automático en los siguientes:

- Encontrar la vista.
- Realizar una acción sobre la vista
- Validar el resultado.

Ilustración 57 Espresso

https://www.guru99.com/testing-tools.html

5.4.4 Watir

Watir debe sus siglas a Web Application Test in Ruby. Fue desarrollado por Bret Pettichord y Paul Rogers en sus inicios, pero como pasa con muchas herramientas al ser abiertas, los participantes en el desarrollo de la misma han ido creciendo con el tiempo.

No es más que un conjunto de bibliotecas bajo la licencia BSD (Berkeley Software Distribution) para el lenguaje de desarrollo Ruby, mediante las cuales se pueden automatizar acciones de los navegadores web.

Trabaja con los principales navegadores web existentes, IE, Firefox, Chrome y Safari.

Ilustración 58 Watir

http://watir.com/

Tiene la capacidad de enlazar con base de datos, leer ficheros de datos y hojas de cálculo, exportar el XML y estructurar los códigos escritos como librerías reutilizables.

Aun teniendo estas capacidades sigue por detrás de Selenium en cuanto a pruebas funcionales en aplicaciones web, ya que el soporte y la información acerca de Selenium es mucho mayor y se ha convertido en la herramienta estándar.

5.5 Ejemplo de prueba funcional

En esta sección nos vamos a entretener un poco más ya que se va a implementar un framework de pruebas partiendo desde cero. Para ello vamos a utilizar la misma plataforma que en las pruebas de rendimiento, pero el entorno va a cambiar ya que aquí no va a afectar tanto el que hagamos estas pruebas, al no realizar llamadas en grandes cantidades. Estas pruebas van a ser lanzadas sobre el entorno de preproducción. Para su implementación utilizaremos diferentes herramientas, las cuales vendrán explicadas en el siguiente apartado. Además de esto nos basaremos en el modelo llamado Page Object Model, el cual introduciremos en apartados posteriores.

5.5.1 Herramientas utilizadas y primeros pasos

Como esta plataforma está desarrollada con AngularJS, vamos a utilizar un framework distinto a los nombrados arriba el cual últimamente está cogiendo mucha fuerza, Protractor.

Ilustración 59 Protractor

https://www.udemy.com/protractor-angular-framework-from-scratch-using-java-nodejs/

Es una herramienta proporcionada por Google la cual nos permite realizar pruebas funcionales automáticas de aplicaciones AngularJS. Utiliza WebDriver para el control de los diferentes navegadores y las pruebas estarán escritas en lenguaje JavaScript.

Además de esto se va a utilizar Cucumber. Es una herramienta utilizada para implementar metodologías BDD (Behaviour Driven Development), mediante la cual se nos permite la ejecución de diferentes descripciones funcionales escritas en texto plano. Estas se escriben en un lenguaje llamado Gherkin, el cual es interpretable por cualquier persona sea cual sea su grado de conocimiento en el campo del desarrollo software. [15]

Ilustración 60 Cucumber

https://mozilla-ni.org/pruebas-de-automatizacion-con-cucumber-bdd-en-equipos-agiles/

Para ello deberemos tener algún editor de textos. En mi caso me he decidido por VisualStudio Code ya que es el que utilizo de forma habitual. Es un editor bastante completo al que se le pueden instalar plugins bastante interesantes.

Ilustración 61 VisualStudio Code

https://es.wikipedia.org/wiki/Visual_Studio_Code

El primer requisito que tenemos que tener en cuenta es la instalación de Java JDK. Lo podremos descargar desde el siguiente enlace:

https://www.oracle.com/technetwork/es/java/javase/downloads/index.html

El siguiente paso para montar este framework es instalar NodeJS, que no es más que un entorno JavaScript que se ejecuta del lado del servidor y está basado en eventos. Desde el siguiente enlace podremos descargárnoslo:

https://nodejs.org/en/download/

Para terminar con la instalación añadiremos el path donde instalemos esta herramienta en nuestras variables de entorno. Una vez lo tenemos, añadido como en la siguiente captura, es posible que necesitemos reiniciar nuestro equipo para comprobar que verdaderamente tenemos instalada la herramienta NodeJS.

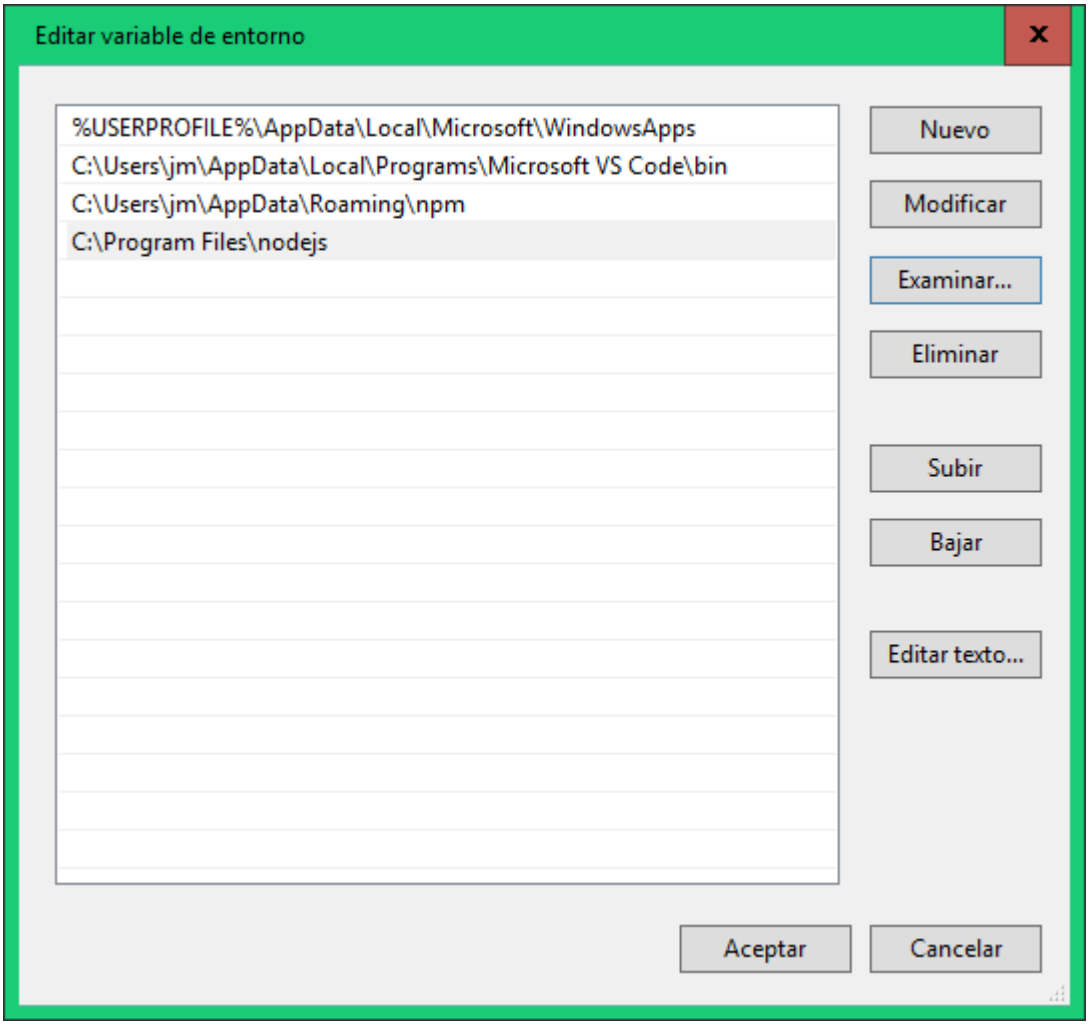


Ilustración 62 Variables entorno

Comprobamos la instalación de NodeJS desde la consola de VisualStudio Code. Para ello sólo tenemos que abrir un terminal y ejecutar el comando node –v:

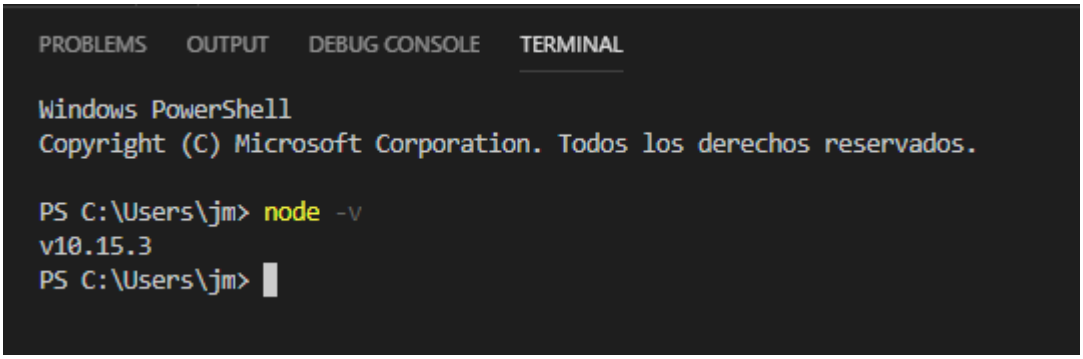


Ilustración 63 Versión NodeJS

Después de comprobar la correcta instalación, comenzaremos con la instalación de todas las

dependencias, entre ellas las que tienen que ver con Protractor y Cucumber.

5.5.2 Dependencias y package.json

Para ello primeramente tendremos que crear nuestro package.json. Este es básicamente un archivo donde se manejan las dependencias de nuestro proyecto. Además también tiene otros parámetros definidos como puede ser la versión, autor,…

Este archivo puede ser creado de forma manual o automática. Creándolo de forma manual los únicos parámetros que son obligatorios son name (nombre) y version (versión). Para crearlo de forma automática nos posicionamos en la ruta raíz del proyecto y ejecutamos npm init.

Aquí se adjunta nuestro fichero package.json con todas las dependencias necesarias para la creación de nuestro framework:

```
{
  "name": "protractor-cucumber-tests",
  "version": "1.0.0",
  "description": "e2e BDD tests",
  "main": "index.js",
  "author": "Jose Manuel Moreno Poveda",
  "dependencies": {
    "base-64": "^0.1.0",
    "chai": "3.5.0",
    "chai-as-promised": "6.0.0",
    "chalk": "^2.3.1",
    "cucumber": "2.3.1",
    "cucumber-html-reporter": "2.0.0",
    "fast-xml-parser": "^3.12.12",
    "fs-extra": "^0.30.0",
    "grunt": "^1.0.1",
    "lodash": "^4.17.11",
    "moment": "^2.22.2",
    "protractor-api-resource": "^1.0.3",
    "transform-runtime": "0.0.0",
    "webdriver-manager": "^12.1.1"
  },
  "scripts": {
    "test": "npm run protractor --",
    "test:multiple": "npm run test -- --capabilities.shardTestFiles=true --
capabilities.maxInstances=5 --params.defaultTimeout=100000",
    "test:wip": "npm run test -- --cucumberOpts.tags @wip",
    "test:debug": "node --inspect ./node_modules/protractor/bin/protractor
conf.js",
    "test:debug-wip": "node --inspect ./node_modules/protractor/bin/protractor
conf.js --cucumberOpts.tags @wip",
    "protractor": "protractor conf.js"
```

```
  },
  "devDependencies": {
    "babel-core": "^6.26.3",
    "babel-plugin-transform-runtime": "^6.23.0",
    "babel-preset-es2015": "^6.24.1",
    "babel-preset-stage-2": "^6.24.1",
    "babel-register": "^6.26.0",
    "grunt-protractor-runner": "5.0.0",
    "husky": "^1.3.1",
    "node-uuid": "^1.4.8",
    "protractor": "^5.1.2",
    "protractor-cucumber-framework": "3.1.2",
    "supertest": "^3.4.2"
  }
}
```

Como se puede comprobar al principio tenemos los parámetros propios del proyecto que descubren su naturaleza. Seguidamente nos encontramos una sección con las dependencias del proyecto, en la que nos encontramos cada una de las dependencias con su correspondiente versión. Esto es de mucha utilidad ya que cualquier usuario mediante la ejecución del comando npm install, se descargará estas dependencias definidas aquí y se instalarán para el correcto funcionamiento del proyecto. A continuación nos encontramos los scripts, que definen los comandos que podemos ejecutar en nuestro proyecto asociándolos a palabras claves que nos facilitan su uso. Por último, nos encontramos con las devDependencies con las diferentes librerías para que el proyecto arranque.

A continuación vamos a descargarnos e instalar todas las dependencias antes descritas en el package.json. Para esto, como antes se mencionó, ejecutaremos el siguiente comando en la ruta de nuestro proyecto:

npm install

Una vez hemos ejecutado ese comando se nos creará la carpeta node modules en la ruta de nuestro proyecto, que albergará todas nuestras dependencias. Y a su vez el archivo package-lock.json, que básicamente sirve de guía para tener las dependencias actualizadas.

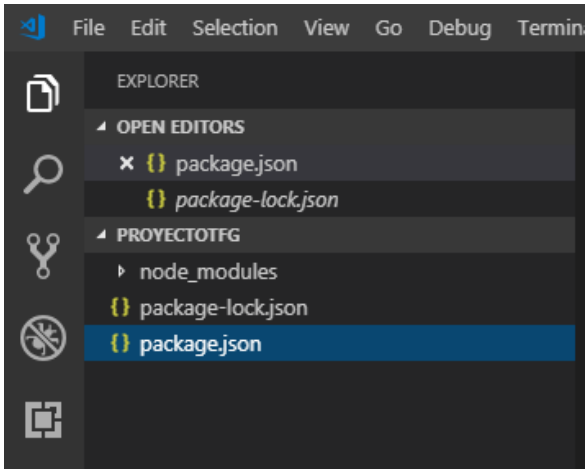


Ilustración 64 Estructura proyecto

Una vez descargadas e instaladas las dependencias, vamos a empezar a configurar nuestro framework. Para ello crearemos un archivo llamado conf.js.

5.5.3 Configuración del framework mediante conf.js

Se va a mostrar este archivo y se van a ir desvelando los principales componentes que lo forman. Se va a mostrar una posible configuración, pero las posibilidades de formar este archivo son enormes, dependiendo de nuestras necesidades y lo que queramos conseguir se podrán definir más o menos parámetros o se podrán utilizar unas configuraciones u otras.

```
var config = require('./config');
exports.config = {
  useAllAngular2AppRoots: true,
  // setting to launch tests directly without selenium server
  directConnect : true,
  // address of running selenium server
  seleniumAddress : 'http://localhost:4444/wd/hub',

  // setting time outs
  getPageTimeOut : 150000,
  allScriptsTimeout : 300000,
  // setting framework
  framework : 'custom',
  frameworkPath : require.resolve('protractor-cucumber-framework'),

  // setting protractor to ignore uncaught exceptions to take care by
protractor-cucumber-framework
  ignoreUncaughtExceptions : true,

  // configuration parameters
```

```
  params: {
    testEnv: 'test',
    defaultTimeout: 15000
    },

  // browser to launch tests
  capabilities : {
    browserName : 'chrome',
    shardTestFiles: false,
    maxInstances: 1,
    chromeOptions : {
      args : [ '--disable-extensions']
    }
  },

  // Specify Test Files
  //
  // Define which tests should execute
  specs : [
        'automatedTest/features/register.feature',
        'automatedTest/features/login.feature'
         ],

  //Define which tests should be excluded from execution.
  exclude : [
      ],

  // Set log level and enables colors for log output
  logLevel : 'verbose',
  coloredLogs : true,

  // arguments to cucumber.js
  cucumberOpts : {
    require : [
                'support/env.js',
                'support/hooks.js',
                'automatedTest/step_definitions/*.js'
    ],
    tags : false,
    format : 'json:reports/cucumber.json',
    profile : false,
    'no-source' : true
  },
  SELENIUM_PROMISE_MANAGER: false,
  onPrepare: function(){
    require('babel-register')
    require('babel-core/register');
```

```
    browser.ignoreSynchronization = true;
    browser.driver.manage().window().maximize();
    browser.get(config.appUrl);
    browser.driver.manage().deleteAllCookies();
  }
};
```

El primer parámetro que nos encontramos, useAllAngular2AppRoots, es bastante importante a la hora de encontrar elementos. Se configura a true para que esta búsqueda de elementos sea bastante más sencilla ya que con esto conseguimos localizar un objeto de diferentes formas, como puede ser mediante selector css, id, xpath,…

Lo siguiente importante que vemos es el parámetro directConnect, también configurado a true. Aunque más abajo podemos ver la dirección para usar el servidor de Selenium, mediante este parámetro configurado a true nos saltamos este paso y directamente corremos nuestras pruebas con los drivers del navagador que configuremos. Esto nos da una mayor velocidad sin embargo para test complejos, como pueden ser test en diferentes idiomas y lanzados en paralelo, es aconsejable el uso de la otra opción que nos dará mejores resultados.

Definimos los timeouts oportunos para lanzar un error y parar la ejecución del test. En este caso podemos ver timeouts para la carga de una página, para esperar a que termine la ejecución de script asíncronos, timeout por defecto para pasarlo como parámetros a las diferentes funciones que definiremos,…

Siguiendo con el archivo, definiremos el framework como custom ya que estamos customizando nuestro propio framework y como se puede observar convivirán Protractor y Cucumber a la hora de desarrollarlo.

Continuando, nos encontramos las capabilities. Tienen que ver con la configuración del navegador en el que queremos que se ejecuten nuestras pruebas. Aquí se ha elegido como navegador de pruebas Chrome. El parámetro de maxInstances describe cuantas instancias del navegador queremos que estén ejecutándose a la vez. Esto es útil a la hora de ejecutar múltiples test en paralelo.

En las secciones spec y exclude, se rellenarán con la localización de los test que queremos que sean ejecutados y los que queremos que sean excluidos respectivamente.

A continuación, un apartado en referencia a los logs. Se configuran dos parámetros uno en referencia a que muestre más información de la que normalmente mostraría y el segundo en referencia a estilos.

Luego nos encontramos configuraciones referentes a Cucumber, como pueden ser la configuración de etiquetas (tags), para etiquetar nuestros tests y ejecutarlos en función de esas etiquetas. El formato de un reporte en formato JSON, que nos dará información de como ha ido la ejecución de cada sentencia de Cucumber.

El parámetro SELENIUM_PROMISE_MANAGER configurado a false nos permite ejecutar acciones asíncronas. La mayoría de nuestras llamadas a funciones serán asíncronas como veremos más adelante. Con esto conseguiremos que hasta que no se ejecuta una sentencia no pase a ejecutar la siguiente.

Lo último que nos encontramos en dicho archivo es la funcionalidad de levantar el navegador, maximizar la ventana en la que se ejecuta el navegador y borrar las cookies para que estas no intervengan en los resultados de los test. En este mismo apartado también nos encontramos sentencias que hacen referencia a babel de las cuales hablaremos más adelante en un archivo específico de esto.

En el apartado params, se puede visualizar el nombre del entorno en el que se van a lanzar las pruebas.

Este vendrá configurado en el siguiente archivo del que vamos a hablar.

5.5.4 Configuración del entorno con archivos .json e index.js

Para ello crearemos la siguiente carpeta de nombre config:

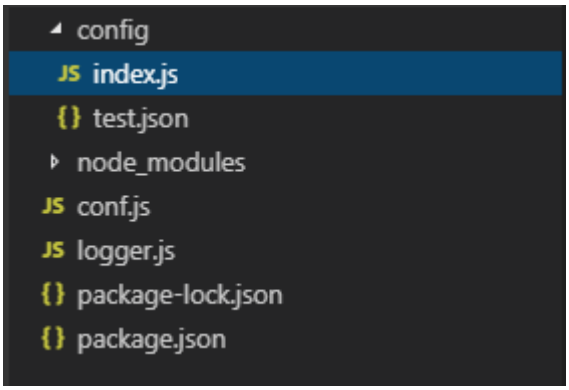


Ilustración 65 Carpeta config

En este caso, por cada entorno en el que queramos lanzar las pruebas, se ha creado un archivo .json con el nombre del entorno, en este caso hemos creado solo el entorno test.json. Este archivo contendrá la URL de este modo:

```
{
    "appUrl": "https://xxx.wanabet.es/bienvenida"
}
```

Así nos será más fácil a la hora de lanzar las pruebas en un entorno u otro, sólo tendremos que cambiar ese parámetro en el archivo de configuración antes expuesto.

Una vez definido nuestro archivo de configuración y los entornos en los que vamos a ir lanzando las pruebas, vamos a ir limando detalles como el tema de los entornos. Antes hemos visto como se definen los entornos en el archivo de configuración y que aspecto tienen los archivos .json que definen estos entornos. El siguiente paso tiene que ver con el fichero que queda en la carpeta config, index.js.

```
const allowedEnvs = ['test'];
const env = process.env.NODE_ENV || 'test';

if (allowedEnvs.indexOf(env) === -1) {
    throw new Error(`NODE_ENV=${env} is not allowed.. options are
${allowedEnvs.join(', ')}`);
}

const config = require(`./${env}.json`);
```

```
module.exports = config;
```

Este archivo index.js no contiene más que la definición de los entornos que tenemos disponibles y en caso de no estar disponible se arroja un error. Como se puede comprobar, solamente tenemos definido nuestro entorno de preproducción, al cual hemos llamado test.

5.5.5 Plugins y presets con babelrc

Una vez configurados nuestros entornos, vamos a pasar a hablar de otro archivo necesario llamado .babelrc, del que antes hemos visto algo en el archivo de configuración.

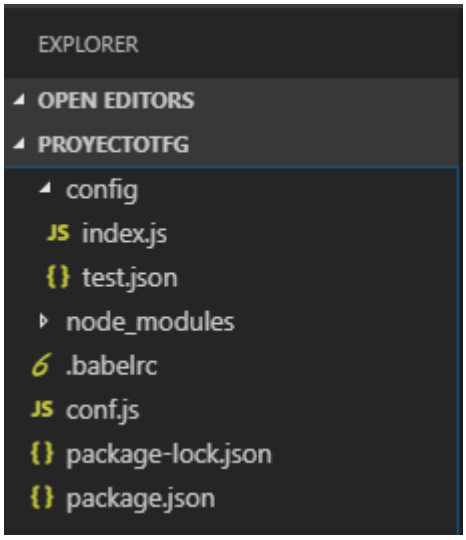


Ilustración 66 Archivo .babelrc

Este archivo configura presets y plugins para la compilación de los tests. Dicho archivo tiene la siguiente forma:

```
{
    "presets": [
      "es2015",
      "stage-3"
    ],
    "plugins": [
      [
        "transform-runtime",
        {
          "polyfill": false,
          "regenerator": true
        }
      ]
```

```
    ]
  }
```

Aquí se deja una URL que sirve de guía para poder configurar tu propio archivo .babelrc:

https://babeljs.io/docs/en/config-files

5.5.6 Automatización de tareas con Gruntfile

El siguiente paso es configurar un archivo llamado Grunfile.

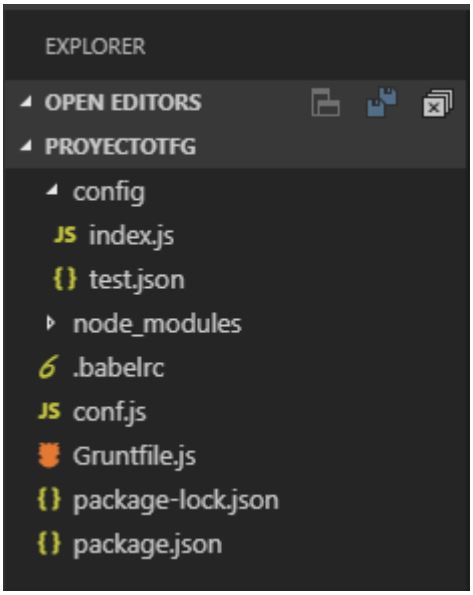


Ilustración 67 Gruntfile.js

Grunt.js es una librería de JavaScript con la cual podemos automatizar ciertas tareas con el fin de ahorrarnos tiempo en diversas tareas como podrían ser el desarrollo y el despliegue de aplicaciones.

En este archivo, Gruntfile.js, vamos a indicarle ciertas tareas a automatizar con un simple comando en formato JSON. Más concretamente, las operaciones en cuestión serán las de ejecutar un test y la de entrar en modo debug. En el código que se muestra a continuación vienen comentados los parámetros más importantes que se han usado:

```
module.exports = function(grunt) {

  // Project configuration.
  grunt.initConfig({
    protractor: {
      test: {
```

```
        options: {
          configFile: "conf.js", // Protractor config file
          keepAlive: false, // If false, the grunt process stops when the test
fails.
          noColor: false, // If true, protractor will not use colors in its
output.
          args: {
            params: {
              testEnv: 'test'
                },
                verbose:false
          }
        }
      },
      debug: {
        options: {
            configFile: "conf.js", // Protractor config file
            keepAlive: true, // If false, the grunt process stops when the test
fails.
            debug: true, // If true, protractor change to debug mode.
            args: {
            params: {
              testEnv: 'test'
               },
               verbose:false
            }
        }
      }
    },
  });

  // Load the plugin that provides the "runner" task.
  grunt.loadNpmTasks('grunt-protractor-runner');

  // Default task(s).
  grunt.registerTask('default', ['runner']);

};
```

JavaScript es un lenguaje de un solo hilo, esto repercute en que solamente se puede ejecutar una tarea en un determinado instante de tiempo. Antes de ejecutar el código, el intérprete de JavaScript entra primeramente en un contexto de ejecución global por defecto. Cada invocación de una función a partir de este punto resultará en la creación de un nuevo contexto de ejecución. Es por eso que crearemos el siguiente archivo.

5.5.7 Contextos de ejecución con context.js

Con este archivo crearemos los contextos de ejecución correspondientes a cada llamada y una vez que se ejecuta esta, se borrará dicho contexto.

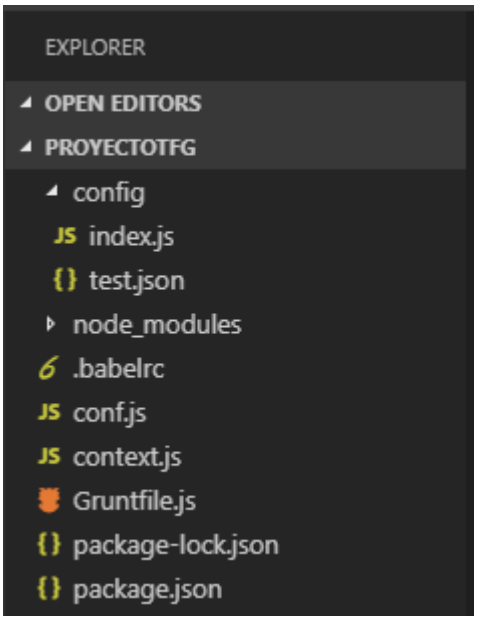


Ilustración 68 Context.js

Aquí se expone el contenido de dicho archivo:

```
let _context = {};

const context = {};

context.get = key => {
  const value = _context[key];
  if (value !== undefined) {
    return value;
  } else {
    throw new Error(key + ' not found in context');
  }
};

context.set = (key, value) => {
  _context[key] = value;
};

context.has = key => {
  return !!_context[key];
};
```

```
context.clear = (key) => {

  if (key) {
    delete _context[key];
  } else {
    _context = {};
  }
};

module.exports = {context};
```

Vamos a seguir con la configuración de nuestro framework definiendo las variables globales que vamos a necesitar, como el timeout, el contexto, EC (Expected Conditions)…

5.5.8 Creación de variables globales con env.js

Para esta tarea crearemos un directorio llamado support en el que estará el siguiente archivo llamado env.js.

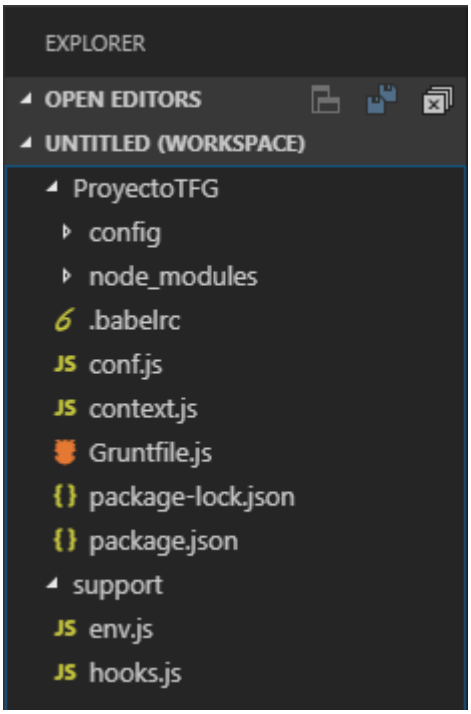


Ilustración 69 Directorio suppport

```
// Environment global variables declaration
var config = require('../config')
var configure = function () {
```

```
    this.setDefaultTimeout(240 * 1000);
};
var timeout = browser.params.defaultTimeout;

const util = require('util');
var chai = require('chai');
var chaiAsPromised = require('chai-as-promised');
var EC = protractor.ExpectedConditions;
//var config;
chai.use(chaiAsPromised);
global.expect = chai.expect;
global.request = require('supertest')(config.apiUrl);
global.idServerRequest = require('supertest')(config.idServerUrl);
global.logger = require('../logger').logger;
global.util = util;
global.EC = EC;
global.config = config;
global.context = require('../context').context;
global.timeout = timeout;
module.exports = configure;
```

5.5.9 Creación de reports mediante hooks.js

En este directorio, además incluiremos otro archivo, hooks.js. Este archivo irá destinado a la creación de los reportes en los diferentes formatos. En estos reportes se informará al usuario de los test que han ido bien, los que han ido mal y los que han sido saltados por cualquier motivo. Además, nos sacará información en un log acerca de porqué ha fallado. También hará una captura en el momento en el que el test falle para facilitarnos asi encontrar los errores y decidir si el error es cosa del software desarrollado o es un falso error.

Tendremos que crear a su vez el directorio donde queremos que se ubiquen las capturas de pantalla y los diferentes reportes en sus diferentes formatos. Para ello crearemos un directorio similar a este:

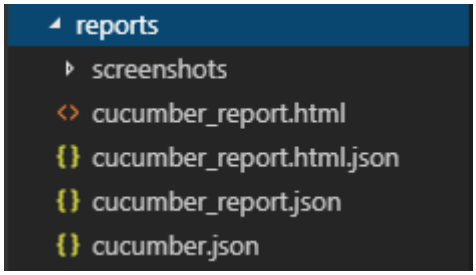


Ilustración 70 Directorio de reportes

```
// Before And After hooks used while feature executes
```

```
const outputDir = './reports/';
const screenshotDir = './reports/screenshots/';
const targetJson = `${outputDir}cucumber_report.json`;

const JsonFormatter = require('cucumber').JsonFormatter;
const fse = require('fs-extra');
const reporter = require('cucumber-html-reporter');

const { defineSupportCode } = require('cucumber');

defineSupportCode(({ setDefaultTimeout }) => {
    setDefaultTimeout(10 * 60 * 1000);
});

defineSupportCode(({ After, registerListener }) => {
    const writeScreenshotToFile = function (image) {
        if (!fse.existsSync(screenshotDir)) {
            fse.mkdirSync(screenshotDir);
        }
        const date = new Date();
        const timestamp = date.getTime();
        const filename = `error_${timestamp}.png`;
        const stream = fse.createWriteStream(screenshotDir + filename);
        stream.write(image);
        stream.end();
    };

    After(function (scenario, done) {
        const self = this;
        if (scenario.isFailed()) {
            browser.takeScreenshot().then((png) => {
                const decodedImage = new
Buffer.from(png.replace(/^data:image\/(png|gif|jpeg);base64,/, ''), 'base64');
                writeScreenshotToFile(decodedImage);
                self.attach(decodedImage, 'image/png');
                done();
                browser.executeScript('window.sessionStorage.clear();');
                browser.executeScript('window.localStorage.clear();');
                browser.get(config.appUrl);
            }, (err) => {
                done(err);
            });
        }
        else {
            done();
            browser.executeScript('window.sessionStorage.clear();');
            browser.executeScript('window.localStorage.clear();');
```

```
            browser.get(config.appUrl);
        }
    });

    const createHtmlReport = function () {
        const options = {
            theme: 'bootstrap',
            jsonDir: './reports/',
            output: `${outputDir}cucumber_report.html`,
            reportSuiteAsScenarios: true,
            launchReport: true
        };
        reporter.generate(options);
    };

    const jsonFormatter = new JsonFormatter();
    jsonFormatter.log = function (string) {
        if (!fse.existsSync(outputDir)) {
            fse.mkdirSync(outputDir);
        }

        fse.writeFile(targetJson, string, (err) => {
            if (err) {
                console.log('Failed to save cucumber test results to json
file.');
                console.log(err);
            } else {
                createHtmlReport(targetJson);
            }
        });
    };

    registerListener(jsonFormatter);
});
```

Antes de meternos de lleno en la creación de los test como tal, vamos a hablar del modelo que vamos a utilizar para implementarlos, que como dijimos en la introducción de esta sección, es el modelo Page Object Model.

5.5.10 Modelo Page Object Model

Este modelo consiste en la creación de un objeto por cada conjunto de elementos significativos de la interfaz con la que el usuario de la plataforma interactúa. Este conjunto de elementos no tiene porqué coincidir con cada una de las páginas por las que vayamos navegando, también se pueden definir objetos por elementos importantes que podamos reutilizar en otros tests.

Con este modelo vamos a conseguir un código robusto y reutilizable. Vamos a poder llamar a los diferentes componentes con sus selectores y sus métodos definidos desde cualquier parte, haciendo un código bastante reutilizable y sencillo de entender.

Una vez definido el modelo en el que nos vamos a basar para implementar nuestros tests, nos vamos a parar a introducir la estructura de estos tests.

5.5.11 Estructura de los tests

Nuestros test van a estar compuestos principalmente por tres tipos de ficheros:

1. **Features:** Este tipo de archivos engloban una serie de escenarios que tienen un factor común, una funcionalidad. Cada escenario es un caso de prueba diferente el cual se divide en steps o pasos para comprobar el funcionamiento correcto del caso de prueba. Estos pasos serán reutilizables y se podrán llamar desde cualquier feature o escenario. Estos archivos estarán escritos en lenguaje Gherkin del cual hemos hablado antes.

2. **Pasos de prueba (Steps definition):** Como su propio nombre indica serán la definición de los pasos que forman los ficheros features. Será un fichero .spec.js, el cual no contendrá lógica alguna. Sólo hará llamadas a los diferentes métodos necesarios.

3. **Pages:** Serán las famosas clases por componentes que hemos nombrado antes. Estos ficheros .po.js contendrán los elementos a interactuar de cada componente importante en nuestros test y sus diferentes métodos, los cuales si albergarán lógica.

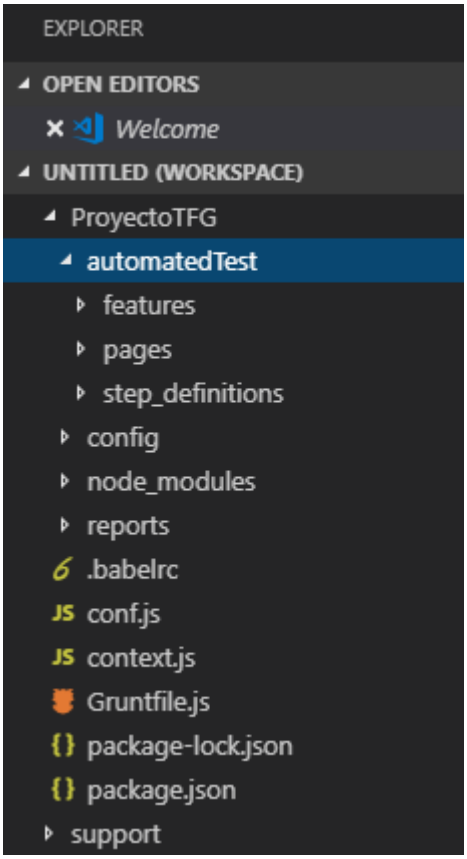


Ilustración 71 Estructura de test

Una vez definida la estructura de nuestras pruebas, vamos a definir las funcionalidades que vamos a cubrir como ejemplo de prueba.

5.5.12 Implementación de los tests

A modo de ejemplo, vamos a implementar el registro de un jugador y el login y logout. Una vez implementados estos, vamos a jugar con estas funcionalidades e implementaremos algunos escenarios más en los que nuestros tests fallen para ver los resultados que obtenemos y comprobarlo en los reportes.

Antes de empezar a crear nuestra primera feature, es conveniente realizar el flujo que queremos implementar de forma manual, para así ver los pasos que tendremos que ir implementando y ver los componentes con los que vamos a interactuar. Esto será útil para la definición de los pasos en nuestra feature y para posteriormente la creación de los diferentes ficheros por componentes importantes en nuestro flujo.

Para el registro nosotros interactuamos con los siguientes componentes:

- La página home o página principal.
- La página en la que el jugador introducirá sus datos personales.
- La página en la que el jugador introducirá sus datos de contacto.
- La página en la que el jugador introducirá sus datos de usuario de la plataforma.

- Un pop-up que salta en diferentes páginas para subir documentos de identidad.
- La página de inicio de sesión.

Una vez que tenemos dividido nuestro flujo de registro en esos componentes, empezaremos creando nuestra feature en referencia al registro. Esta contendrá un escenario con todos los pasos a seguir para realizar un registro en nuestra plataforma. Aquí se deja nuestra feature implementada:

```
Feature: User register in Wanabet platform
    As a user
    I want to be able to register in wanabet platform

    Background: Login background steps
        Given User is on Wanabet portal

    Scenario: Register one user in Wanabet platform
        Given User select register button
        When User enters personal details
            | name   | surname1 | surname2 | day    | month | year   |
nationality  | documentNumber    | fiscalRegion  |
            | Driss  | García   |          | 23     | enero | 1966   | El
Salvador  | X6246189X         | La Rioja      |
            And select Next
            And select Continue in the pop-up
            And User enters contact details
            | phone     | mail                  | address               | CP
| province      | location      |
            | 666666666 | driss@mailinator.com  | Carlos Lopez Bustos   | 13003
| Jaén          | Jaén          |
            And select Next
            And User enters user details
            | username  | password  |  bono         |
            | dristest  | qwer1234. | No quiero bono|
            And User authorisations data
            And User confirms age
            And User select submit
            And select cancel upload documentation
        Then User is in init session screen
```

Aquí tendremos la definición de la feature cuya funcionalidad se basa en el registro de jugadores. También tenemos implementado un escenario en el que registraremos un usuario correctamente con cada uno de sus pasos en lenguaje Gherkins, en el que Given será una precondición, When hará referencia a acciones a realizar y Then, al resultado esperado. Como podemos comprobar también pasamos datos para rellenar los campos del registro en forma de tabla que más tarde trataremos tanto en la definición de cada paso como en los pages. El punto de partida es lo que se denomina

Background, el cual en nuestro caso es que estamos en la página principal de nuestro portal Wanabet.

El siguiente paso será la implementación de los pasos de la feature que hemos desarrollado. Para ello tendremos que irnos creando ficheros .spec.js como los que definimos anteriormente, los cuales contendrán la implementación de estos pasos.

Al trabajar con el patrón Page Object Model, por cada componente importante en nuestra aplicación se irán creando diferentes ficheros. Estos ficheros, recogerán los pasos que interactúan con cada uno de dichos componentes importantes para la realización del flujo del registro en este caso. Con los cual empezaremos por crearnos nuestros 6 ficheros, uno por cada componente que analizamos anteriormente.

El primer fichero a desarrollar será el que tiene que ver con el primer componente, home.spec.js. Aquí las primeras funcionalidades a implementar serán las correspondientes al Background y el seleccionar el botón de registrar.

```
const { defineSupportCode } = require('cucumber');
// Step definitions
const homePage = require('../pages/homePage.po');
const config = require('../../config');
const helperButtons = require('../utils/buttons.js');

defineSupportCode(({ Given, When, Then }) => {

    Given('User is on Wanabet portal', async () => {
        await homePage.waitforWanabetIcon();
    });

    Given('User select register button', async () => {
        await helperButtons.clickRegisterButton();
    });
});
```

Como se puede ver, en el fichero están definidas las dos funcionalidades que vamos a usar de este componente. La primera, es una simple comprobación de que estamos en el punto de partida. Esto se hará comprobando la visibilidad del icono principal de nuestro portal. La segunda, se encargará de hacer clic en el botón registrar.

Vamos a implementar dichas funciones definidas en nuestro archivo .spec.js. Para ello tendremos que crearnos nuestro fichero page correspondiente, home.po.js:

```
const helperSelectors = require('../utils/selectors.js');

const HomePage = function () {
    // Element definition
    this.wanabetIcon = element(by.css('.header__logo > a:nth-child(1) > img:nth-
child(2)'));
```

```
    this.waitforWanabetIcon = async function () {
        await helperSelectors.waitPresenceOf(this.wanabetIcon,timeout);
    };
};
module.exports = new HomePage();
```

Aquí podemos ver la definición de la primera funcionalidad, esperar a que el icono de la página principal esté visible. Para ello tenemos que coger el identificador del icono, el cual en este caso ha sido mediante css ya que no disponía de id. Lo recomendable sería obtener el elemento mediante id ya que es un identificador único del elemento, mientras que con css o xpath en cuanto se cambie la estructura de la página, va a cambiar nuestro css y no se localizará el componente con lo cual nuestro test fallará.

Una vez tenemos el elemento, implementamos el método waitforWanabetIcon, en el que hacemos una llamada a un método del archivo selectors.js llamado, waitPresenceOf, en el que se comprobará la presencia del elemento obtenido en un periodo de tiempo no superior al timeout por defecto definido en el archivo de configuración. En este archivo se han incluido funciones las cuales vamos a utilizar muy frecuentemente. Son funciones propias de protractor las cuales han sido mejoradas a la hora de mostrar trazabilidad en los logs y a la hora de realizar ciertas comprobaciones, como por ejemplo, el comprobar la presencia, visibilidad y que un elemento es clicable antes de hacer ese clic. Aquí se adjunta dicho fichero el cual hemos incorporado en una carpeta llamada utils:

```
const moment = require('moment');

const HelperSelectors = function () {

    this.sureClick = async function (clickId) {
        const elementToClickIsPresent = EC.presenceOf(clickId);
        const elementToClickIsVisible = EC.visibilityOf(clickId);
        const elementToClickIsClickable = EC.elementToBeClickable(clickId);
        await browser.wait(
            EC.and(
                elementToClickIsPresent,
                elementToClickIsVisible,
                elementToClickIsClickable
            ),
            timeout,
        );
        let GotAStaleElement = true;
        while (GotAStaleElement) {
            GotAStaleElement = false;
            try {
                await clickId.click();
            } catch (StaleElementReferenceException) {
```

```
                await console.log(`Notification: Found ${clickId} as a stale
element, will try to click again...`);
                GotAStaleElement = true;
            }
        }
    };

    this.waitPresenceOf = async function (pageElement, timeout) {
        await browser.wait(EC.presenceOf(pageElement), timeout, `Error waiting
the presence of ${pageElement}`);
    };

    this.waitVisibilityOf = async function (pageElement, timeout) {
        await browser.wait(EC.visibilityOf(pageElement), timeout, `Error waiting
the visibility of ${pageElement}`);
    };

    this.waitInvisibilityOf = async function (pageElement, timeout) {
        await browser.wait(EC.invisibilityOf(pageElement), timeout, `Error
waiting the invisibility of ${pageElement}`);
    };

    this.waitToBeClickable = async function (pageElement, timeout) {
        await browser.wait(EC.elementToBeClickable(pageElement), timeout, `Error
waiting ${pageElement} to be clickable`);
    };

    this.waitTextInElement = async function (pageElement, text, timeout) {
        await browser.wait(
            EC.textToBePresentInElement(pageElement, text),
            timeout,
            `Error waiting text present in ${pageElement}`
        );
    };

    this.getFirstHashRow = function (table) {
        return table.hashes()[0];
    };
};

module.exports = new HelperSelectors();
```

La siguiente funcionalidad a implementar era la de hacer clic al botón de registrar un jugador. Como se puede comprobar, se llama a un método del fichero buttons.js. Este fichero contendrá la definición de todos los botones que nos iremos encontrando en nuestro flujo. Se ha creado por comodidad ya que dichos botones aparecerán en más páginas y es una forma útil de reutilizar código. Este fichero tiene

la siguiente forma:

```
const HelperSelectors = require('./selectors.js');

const HelperButtons = function () {
    this.registerButton = element(by.css('a.btn'));

    // Some buttons
    this.clickRegisterButton = async function () {
        await HelperSelectors.sureClick(this.registerButton);
    };
};

module.exports = new HelperButtons();
```

Como podemos comprobar se obtiene el botón mediante css y se llama al método del archivo selectors.js, en el que se hará un clic después de comprobar que diferentes propiedades de este.

Una vez implementado todo lo referente a la página principal, procederemos a hacer lo mismo con los diferentes componentes. En este caso el siguiente componente es la página en la que se introducen los datos personales. El primer paso, como con el anterior, es definir las diferentes funcionalidades que se requieren de este componente. Las funcionalidades serán rellenar los datos personales, hacer clic en el botón siguiente y hacer clic en el botón continuar de un pop-up.

Las dos últimas funcionalidades en referencia a hacer clic a botones se implementarán en el fichero buttons.js, al igual que la que vimos anteriormente.

La primera funcionalidad, la cual consiste en rellenar los diferentes datos personales del usuario, llamará a una función del fichero selectors.js la cual se encargará de coger los datos que le hemos pasado a través del register.feature a modo de tabla. Una vez que tenemos estos datos, iremos pasándoselos mediante llamadas a funciones implementadas en el page correspondiente, para así ir rellenando campo a campo.

```
const { defineSupportCode } = require('cucumber');
// Step definitions
const personalDataPage = require('../pages/personalDataPage.po');
const config = require('../../config');
const helperButtons = require('../utils/buttons.js');
const helperSelectors = require('../utils/selectors.js');


defineSupportCode(({ Given, When, Then }) => {
  When('User enters personal details', async (personalDataTable) => {
        const data = helperSelectors.getFirstHashRow(personalDataTable);

    await personalDataPage.setName(data.name);
        await personalDataPage.setSurname1(data.surname1);
```

```
        await personalDataPage.setSurname2(data.surname2);
        await personalDataPage.selectMale();
        await personalDataPage.setDay(data.day);
        await personalDataPage.setMonth(data.month);
        await personalDataPage.setYear(data.year);
        await personalDataPage.setNationality(data.nationality);
        await personalDataPage.setdocumentNumber(data.documentNumber);
        await personalDataPage.setfiscalRegion(data.fiscalRegion);
    });

    When('select Next', async () => {
    await helperButtons.clickNextButton();
    });

    When('select Continue in the pop-up', async () => {
    await helperButtons.clickContinuePopupButton();
    });
});
```

Y aquí el Page correspondiente:

```
const helperSelectors = require('../utils/selectors.js');

const PersonalDataPage = function () {
    // Element definition
    this.name = element(by.css('div.form-group:nth-child(3) > input:nth-
child(1)'));
    this.surname1 = element(by.css('div.form-group:nth-child(4) > input:nth-
child(1)'));
    this.surname2 = element(by.css('div.form-group:nth-child(5) > input:nth-
child(1)'));
    this.genderMale = element(by.css('label.btn:nth-child(1)'));
    this.genderFemale = element(by.css('label.btn:nth-child(2)'));
    this.day = element(by.css('div.form-row:nth-child(2) > div:nth-child(1) >
select:nth-child(1)'));
    this.month = element(by.css('.col-5 > select:nth-child(1)'));
    this.year = element(by.css('.col-4 > select:nth-child(1)'));
    this.nationality = element(by.css('div.form-group:nth-child(8) > select:nth-
child(1)'));
    this.documentNumber = element(by.css('div.form-row div.col-9 input.form-
control.ng-untouched.ng-pristine.ng-invalid'));
    this.fiscalRegion = element(by.css('select.ng-invalid'));

    this.setName = async function (nameStr) {
        await this.name.sendKeys(nameStr);
    };
```

```
    this.setSurname1 = async function (surname1Str) {
        await this.surname1.sendKeys(surname1Str);
    };

    this.setSurname2 = async function (surname2Str) {
        await this.surname2.sendKeys(surname2Str);
    };

    this.selectMale = async function () {
        await helperSelectors.sureClick(this.genderMale);
    };

    this.setDay = async function (dayStr) {
        await this.day.sendKeys(dayStr);
    };

    this.setMonth = async function (monthStr) {
        await this.month.sendKeys(monthStr);
    };

    this.setYear = async function (yearStr) {
        await this.year.sendKeys(yearStr);
    };

    this.setNationality = async function (nationalityStr) {
        await this.nationality.sendKeys(nationalityStr);
    };

    this.setdocumentNumber = async function (documentNumberStr) {
        browser.sleep(1000);
        await this.documentNumber.sendKeys(documentNumberStr);
    };

    this.setfiscalRegion = async function (fiscalRegionStr) {
        await this.fiscalRegion.sendKeys(fiscalRegionStr);
    };
};
module.exports = new PersonalDataPage();
```

Mediante la función sendKeys, mandamos el dato al campo obtenido mediante css.

El siguiente componente en el que introducimos los datos de contacto haremos lo mismo que en este. Tendrá la misma funcionalidad de rellenar los datos y de hacer clic en el botón siguiente.

El tercer componente también será similar, ya que tendremos que rellenar los datos de usuario. Esta vez en igual de hacer clic en el botón siguiente, tendremos que hacerlo en el botón de completar o submit. Previamente a esto, tendremos que aceptar los términos y condiciones y también confirmar

que tenemos más de 18 años, para esto haremos clic en dos checkboxs. Aquí se deja la definición de los pasos y el page correspondiente:

```
const { defineSupportCode } = require('cucumber');
// Step definitions
const userDataPage = require('../pages/userDataPage.po');
const config = require('../../config');
const helperButtons = require('../utils/buttons.js');
const helperSelectors = require('../utils/selectors.js');


defineSupportCode(({ Given, When, Then }) => {
    When('User enters user details', async (userDataTable) => {
        const data = helperSelectors.getFirstHashRow(userDataTable);

        await userDataPage.setUsername(data.username);
        await userDataPage.setPassword(data.password);
        await userDataPage.setBono(data.bono);
    });

    When('User authorisations data', async () => {
        await userDataPage.authorisationData();
    });

    When('User confirms age', async () => {
        await userDataPage.confirmAge();
    });

    When('User select submit', async () => {
        await helperButtons.clickSubmitButton();
    });

});
```

Como comprobaremos a continuación siguen siendo las mismas funciones de enviar datos y hacer clics, sólo que está vez los elementos son checkboxs.

```
const helperSelectors = require('../utils/selectors.js');

const UserDataPage = function () {
    // Element definition
    this.username = element(by.css('div.form-group:nth-child(2) > input:nth-
child(1)'));
    this.password = element(by.css('div.form-group:nth-child(3) > input:nth-
child(1)'));
```

```
    this.bono = element(by.css('select.form-control'));
    this.authorisationDataCheck = element(by.css('div.form-group:nth-child(6) >
label:nth-child(2)'));
    this.confirmAgeCheck = element(by.css('div.form-group:nth-child(7) >
label:nth-child(2)'));

    this.setUsername = async function (usernameStr) {
        await browser.sleep(1000);
        await this.username.sendKeys(usernameStr);
    };

    this.setPassword = async function (passwordStr) {
        await this.password.sendKeys(passwordStr);
    };

    this.setBono = async function (bonoStr) {
        await this.bono.sendKeys(bonoStr);
    };

    this.authorisationData = async function () {
        await helperSelectors.sureClick(this.authorisationDataCheck);
    };

    this.confirmAge = async function () {
        await helperSelectors.sureClick(this.confirmAgeCheck);
    };
};
module.exports = new UserDataPage();
```

El cuarto componente sería el del pop-up. En este tendríamos que implementar otra funcionalidad igual que las anteriores, hacer clic al botón continuar, con lo cual no se va a desarrollar.

El último componente es el de la página de bienvenida. Aquí daremos por bueno el test si se comprueba correctamente el mensaje de bienvenida una vez finalizado el proceso de registro. Vamos a implementar la funcionalidad en el spec.js:

```
const { defineSupportCode } = require('cucumber');
// Step definitions
const loginPage = require('../pages/loginPage.po');
const config = require('../../config');

defineSupportCode(({ Given, When, Then }) => {
    When('User is in init session screen', async () => {
        await loginPage.waitforWanabetLogin();
        });
});
```

Como vemos llama a una función del fichero page correspondiente. En este fichero page se llamará a una función para la comprobación del texto de bienvenida en el componente correspondiente.

```
const helperSelectors = require('../utils/selectors.js');

const LoginPage = function () {
    // Element definition
    this.wanabetLoginHead = element(by.css('.messages-page__title > p:nth-
child(1)'));

    this.waitforWanabetLogin = async function () {
        await helperSelectors.waitTextInElement(this.wanabetLoginHead,'Gracias
por unirte a nosotros,',timeout);
    };
};
module.exports = new LoginPage();
```

Como podemos ver, se obtiene el elemento como en los casos anteriores mediante css y se llama a la función del fichero selectors.js encargada de la comprobación de textos en elementos. A esta función se le pasa el componente en donde buscar el texto, el texto y el timeout por defecto definido en el fichero de configuración.

Con esto ya tendríamos nuestro primer test montado por completo. A continuación vamos a implementar el test de login de un jugador.

```
Feature: User login in Wanabet platform
    As a user
    I want to be able to logout in wanabet platform

    Background: Login background steps
        Given User is on Wanabet portal

    Scenario: Login in Wanabet platform
        Given User select login button
        When User enters login data
            | user      | pass      |
            | dristest  | qwer1234. |
            And select access
            And select cancel upload documentation
            And select user icon
            And User select close session
        Then User session is close
```

Las funciones a implementar para el test del login son iguales que las anteriores, se basan en hacer clics en componentes y enviar datos. Además, se realizar todo desde la página principal con lo cual no tendremos que crear ningún fichero nuevo, habrá que ir completando los archivos antes definidos.

5.5.13 Pintar el log

Como complemento final para que sean más legibles los logs que arrojan las ejecuciones de las pruebas, podemos crear el siguiente archivo, al cual lo hemos llamado logger.js. Este archivo básicamente lo que hará será darle forma al log mediante estilos. Un ejemplo, que se ve claramente aquí, es que los errores los pinta en color rojo para que sean más intuitivos. Esto es a gusto del usuario pero es un archivo bastante útil a la hora de depurar código.

```
const chalk = require('chalk');

const logger = {};

const logLevel = process.env.LOG_LEVEL || 1;

logger.error    = message => writeLog(message, 'red');
logger.success  = message => writeLog(message, 'greenBright');

logger.info     = message => {
  if (logLevel > 1) {
    writeLog(message, 'blueBright');
  }
}

logger.trace = message => {
  if (logLevel > 2) {
    writeLog(message, 'yellow');
  }
}

console.log(`
` +
chalk.bgBlue.bold.black('**********************************************************
*****************************************') + `
` + chalk.blueBright('     LOG_LEVEL: ') + chalk.white(logLevel) + `
` + chalk.red('         error: ') + chalk.white('enabled') + `
` + chalk.greenBright('       success: ') + chalk.white('enabled') + `
` + (logLevel > 1 ? chalk.blueBright('          info: ') +
chalk.white('enabled') : chalk.grey('          info: ') +
chalk.grey('disabled')) + `
` + (logLevel > 2 ? chalk.yellow('         trace: ') + chalk.white('enabled') :
chalk.grey('         trace: ') + chalk.grey('disabled')) + `
```

```
` +
chalk.bgBlue.bold.black('**************************************************
*****************************************') + `
`);

export {logger};

function writeLog(message, color) {
  if (typeof message === 'object') {
    message = JSON.stringify(message);
  }
  if (color) {
    message = chalk[color](message);
  }
  console.log(message);
}
```

5.5.14 Ejecución de los tests

Una vez tenemos nuestros dos tests implementados y nuestro framework montado, vamos a ejecutar estos dos para ver el resultado arrojado. Primeramente ejecutaremos de forma secuencial el registro y el login, los cuales nos deben arrojar resultados positivos. Configuramos en nuestro archivo de configuración la ejecución de estos tests, abrimos un nuevo terminal y ejecutamos el siguiente comando:

Ilustración 72 Ejecución de los tests

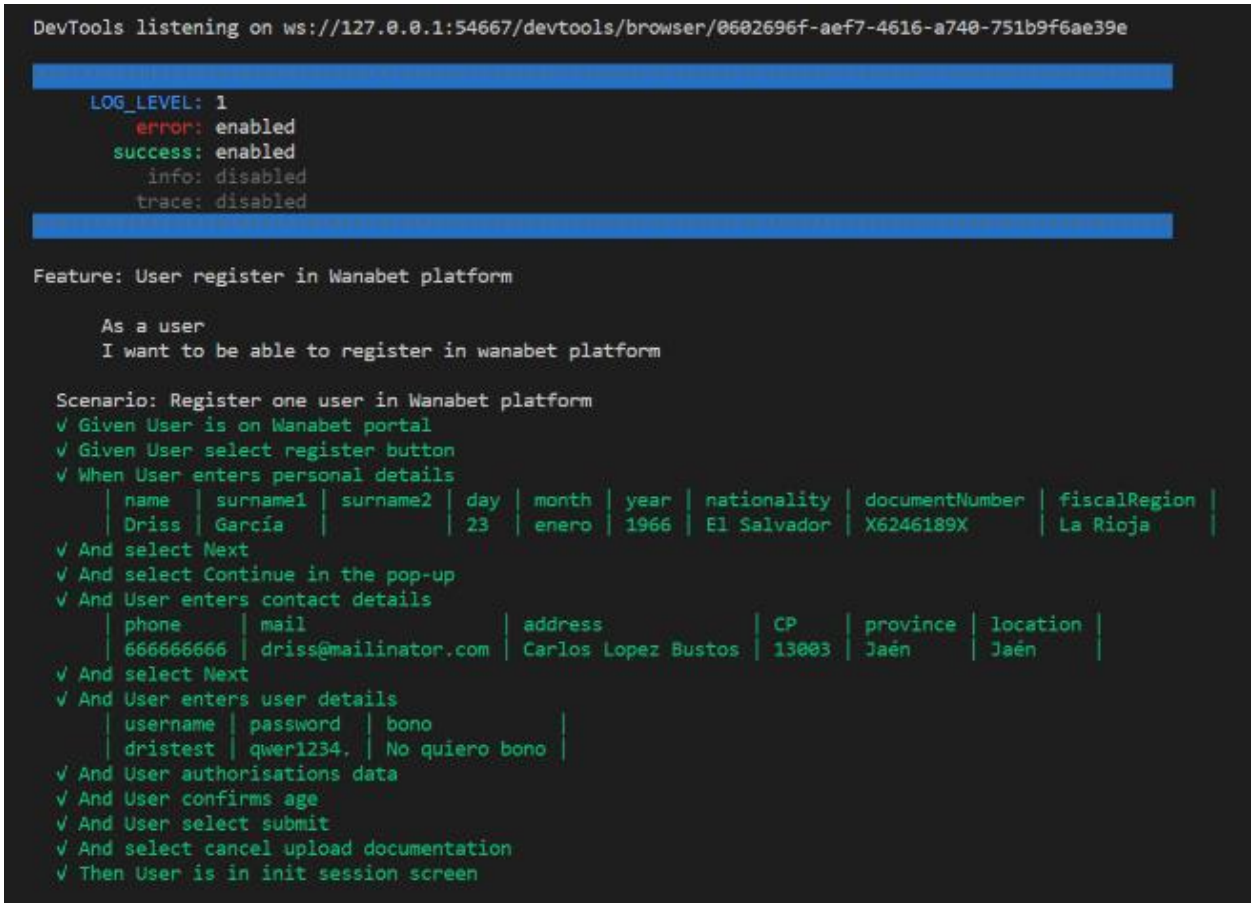


Ilustración 73 Ejecución del registro

Como podemos comprobar van saliendo todos los pasos implementados en nuestras features y en este caso al arrojar un resultado positivo, se muestran en color verde ya que lo configuramos así con anterioridad. Seguidamente a esta register.feature se ejecuta la del login que tendremos los resultados aquí abajo:

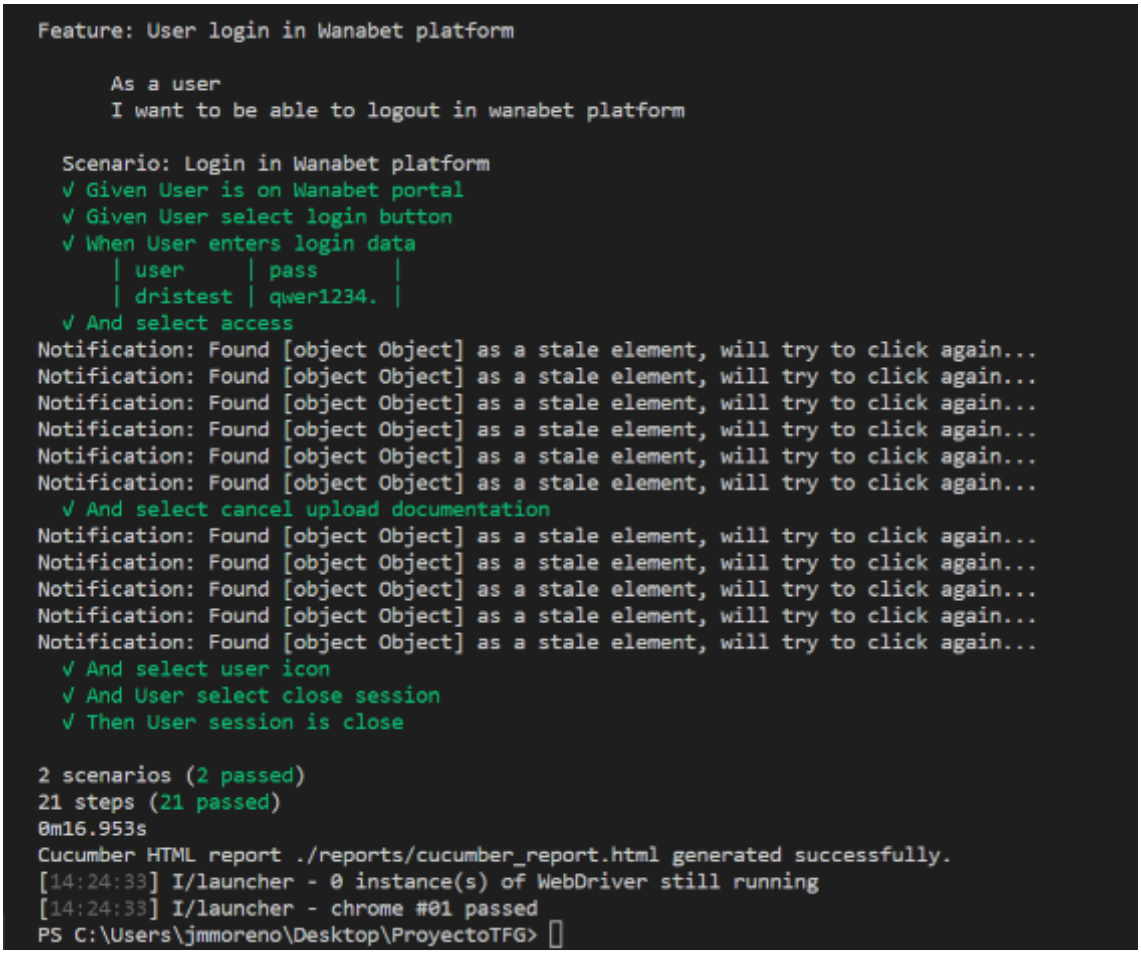


Ilustración 74 Ejecución del login

Como podemos comprobar como en la anterior, también se nos muestran todos los pasos implementados en color verde, lo que supone que los dos tests han acabado con resultado positivo como pone al final de nuestro log, 2 escenarios pasados y un total de 21 pasos ejecutados con éxito.

A parte de esto, podemos comprobar que se han generado nuestros reportes de forma correcta en formtato html, en la ruta indicada en nuestro archivo hooks.js.

Antes de abrir estos reportes vamos a ejecutar una vez más el escenario del registro de un usuario. Lo vamos a hacer con los mismos datos del usuario que ya registramos anteriormente, con lo cual nos tiene que dar un error desde la plataforma y no nos debe registrar el nuevo usuario con los mismos datos.

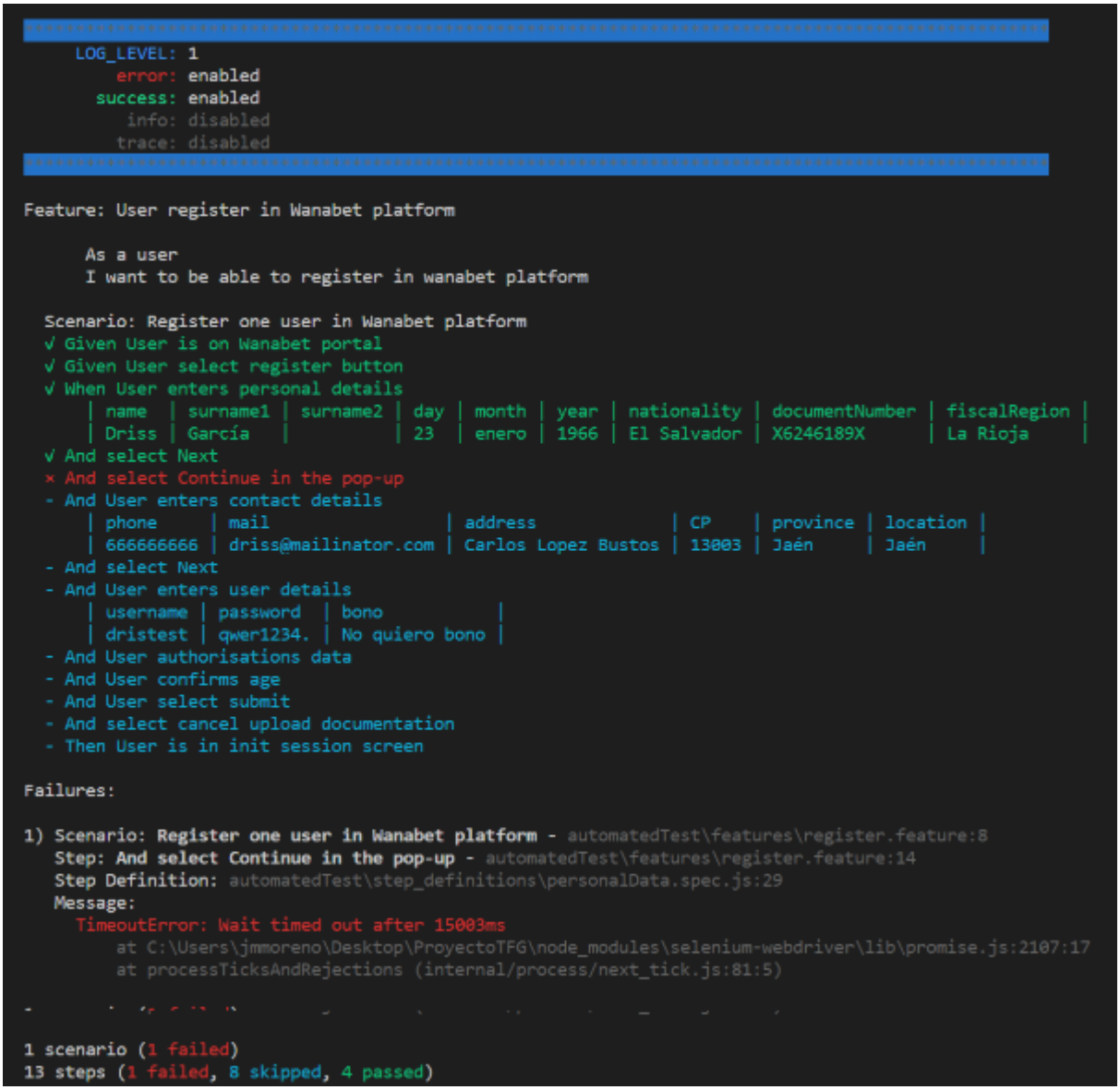


Ilustración 75 Segunda ejecución del registro

Este test falla como esperábamos con el propósito de ver que pasa en caso de un test fallido y ver más datos en los reportes generados. Pasa el paso de rellenar los campos pero al hacer clic en el botón siguiente no sigue con la ejecución de la siguiente frase, ya que el pop-up no ha aparecido y por tanto no ha conseguido localizar el siguiente elemento en el que hacer clic. En cuanto un paso resulta erróneo, los demás pasos se saltan y la ejecución del test acaba siendo fallida.

Ahora es cuando vamos a abrir el reporte html para ver concretamente la captura de donde ha fallado para así poder ver si se trata de un error de la plataforma o de un error nuestro, como es este caso ya que hicimos fallar a propósito esta ejecución.

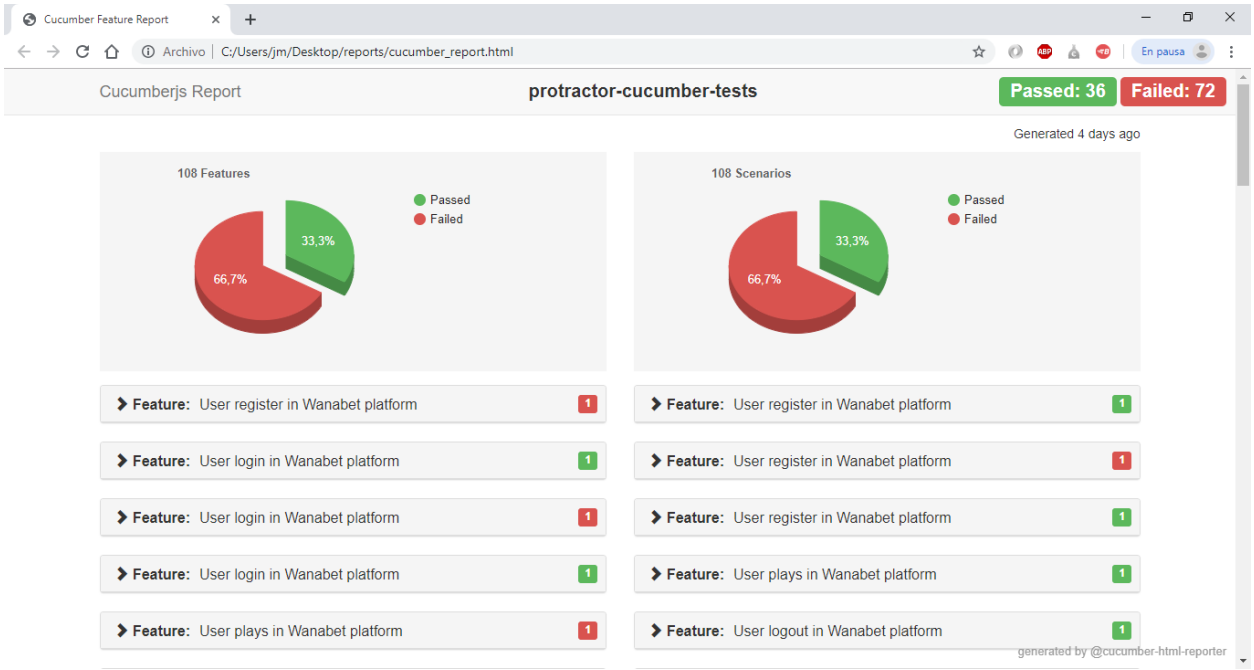


Ilustración 76 Reporte html

Podemos ver todas las ejecuciones de nuestros tests configurados con un gráfico en el que se nos muestran por separado tanto nuestras features con éxito y con fracaso, como nuestros escenarios con éxito o fracaso. Esto nos ayudará a saber como de estable es nuestra aplicación a simple vista.

Si nos adentramos dentro de uno de los test que han ido correctamente podremos ver un resumen de nuestros pasos ejecutados con éxito y a su lado el tiempo que se ha tardado en ejecutar dicho paso.

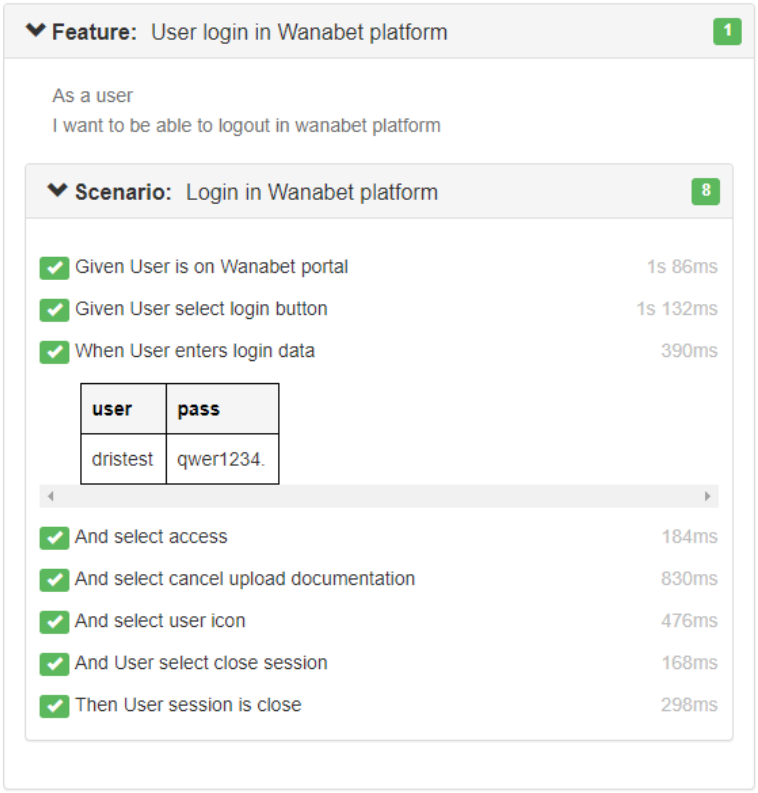


Ilustración 77 Reporte de test con éxito

Por contraposición, vamos a ver ahora que en un test fallido se nos muestran los pasos con éxito como el anterior caso, el paso en el que el test fallo en color rojo y los demás pasos no ejecutados en color amarillo.

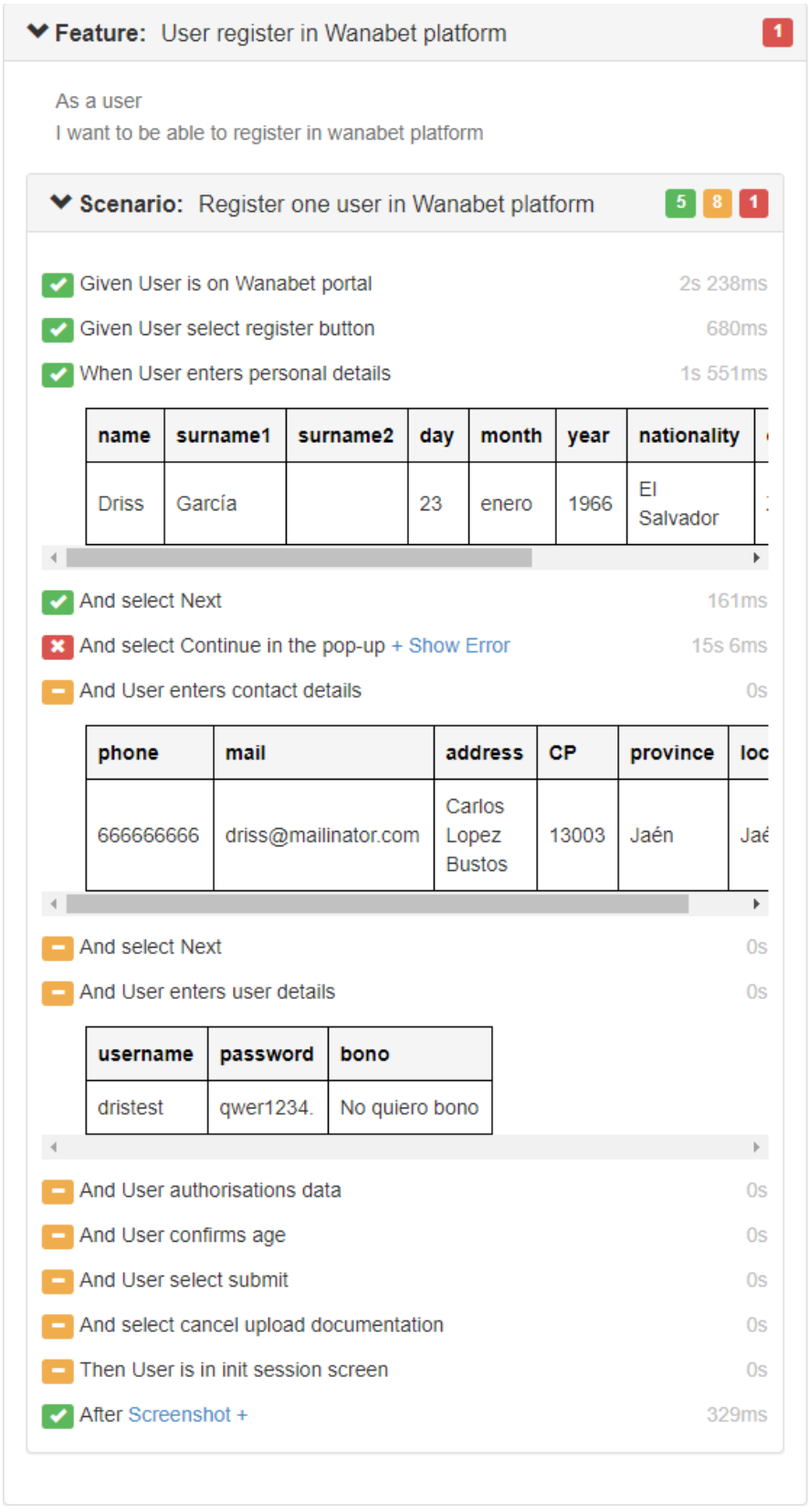


Ilustración 78 Reporte de test fallido

Es muy interesante ver que al lado del paso fallido encontramos una pequeña traza acerca de por qué

ha fallado nuestro test:

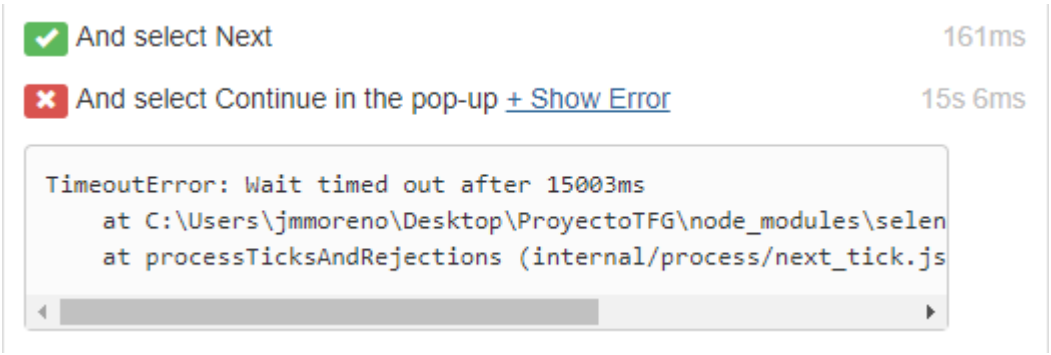


Ilustración 79 Traza del fallo

Por último, también nos muestra una captura de pantalla en el momento del fallo para hacernos más intuitiva la búsqueda del error y saber la causa de ese fallo:

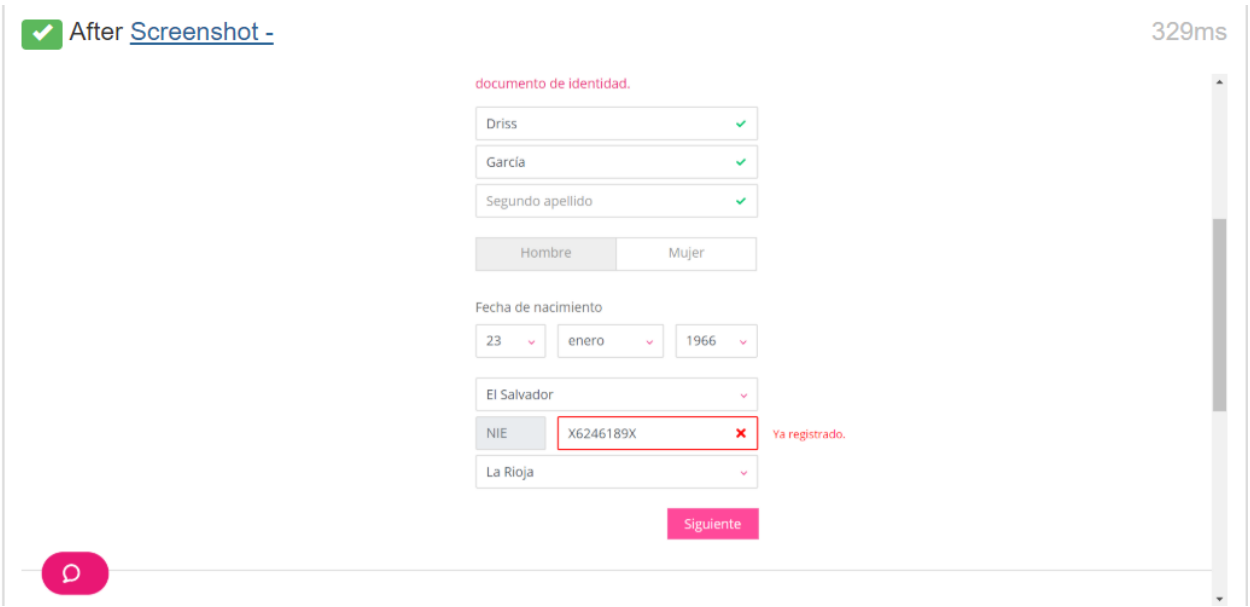


Ilustración 80 Captura del fallo

Con esto último podemos dar por la implementación de este framework de pruebas automáticas funcionales con Protractor.

6 CONCLUSIONES

6.1 Conclusiones finales

Este trabajo de fin de grado pone en evidencia la necesidad de tareas de calidad dentro del desarrollo de cualquier aplicación software. Aunque es cierto que realizar dichas tareas durante el ciclo de desarrollo no evita posibles errores en nuestras aplicaciones, si que los va a minimizar y el impacto negativo será mucho menor.

Como hemos podido comprobar todos y cada uno de los tipos de pruebas, desarrollados en este trabajo, tienen su importancia y ninguno puede considerarse más importante que otro. El desarrollo de estos tipos de pruebas dependerá del tipo de aplicación que queramos crear, los usuarios a los que va a dirigirse, los recursos de los que se dispondrán… No podemos elegir entre funcionalidad, seguridad, rendimiento… Con esto se quiere exponer que el resultado óptimo sería desarrollar pruebas en todos los ámbitos antes descritos con el objetivo de crear una aplicación estable, segura y con funcionalidades interesantes para sus usuarios, pero siendo realista y teniendo en cuenta todos nuestros recursos y el alcance de esta.

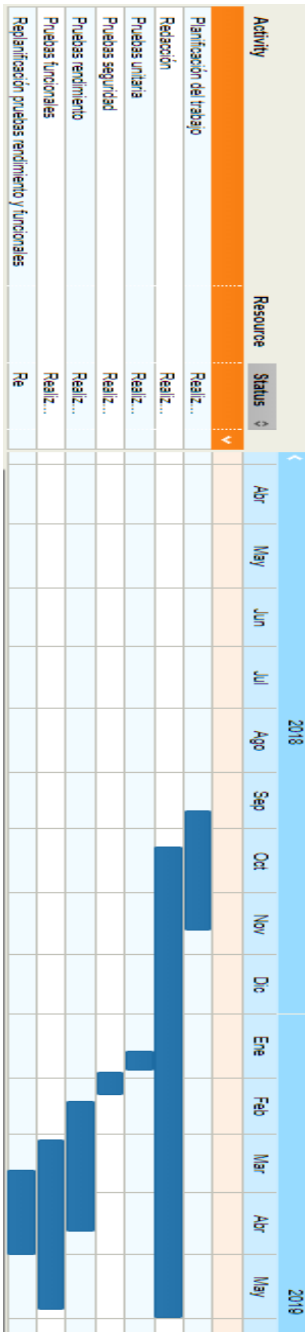
Con este trabajo quiero promover la calidad, tanto a nivel personal como a nivel empresarial, ya que todo trabajo desarrollador tendrá fallos, los desarrolladores son humanos. Un buen desarrollo combinado con un buen proceso de calidad puede favorecer una diferenciación con el resto de nuestros competidores y escalar a un nivel superior en el mercado. Esto será bien recibido por el usuario, el cual comparará nuestro producto con el de otros competidores. Con lo cual puede ser un buen paso para que nuestros desarrollos adquieran cierto nivel y conseguir objetivos más ambiciosos.

6.2 Diagrama de Gantt

El diagrama de Gantt es una herramienta para la organización y la planificación de ciertas tareas a lo largo de un periodo de tiempo determinado. Este permite realizar un seguimiento de dichas tareas y controlar como se avanza a lo largo de un proyecto.

El diagrama muestras las tareas representadas por barras horizontales a lo largo del eje temporal. Estas tareas pueden solaparse.

Aquí se adjunta el diagrama de Gantt de este proyecto:



REFERENCIAS

[1] F. Alonso, L. Martínez y F. J. Segovia, Introducción a la Ingeniería del Software, Modelos de desarrollo de programas, vol. 2, Madrid: Delta, 2005.

[2] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Aplicaci%C3%B3n_web.

[3] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Aplicaci%C3%B3n_m%C3%B3vil.

[4] «Alegsa,» [En línea]. Available: http://www.alegsa.com.ar/Dic/aplicacion_movil.php.

[5] «Yeeply,» [En línea]. Available: https://www.yeeply.com/blog/6-tipos-desarrollo-de-aplicaciones-web/.

[6] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Pruebas_de_software.

[7] NUnit. [En línea]. Available: https://nunit.org/.

[8] Guru99. [En línea]. Available: https://www.guru99.com/what-is-security-testing.html.

[9] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/WebSocket.

[10] «Guru99,» [En línea]. Available: https://www.guru99.com/performance-testing.html.

[11] «Guru99,» [En línea]. Available: https://www.guru99.com/introduction-to-jmeter.html.

[12] «Globe testing,» [En línea]. Available: https://www.globetesting.com/2012/01/novedades-en-hp-loadrunner-11/.

[13] «Proyectos Agiles,» [En línea]. Available: https://proyectosagiles.org/facilitador-scrum-master/.

[14] «Kibernum,» [En línea]. Available: http://www.kibernum.com/noticias/por-que-son-importantes-las-pruebas-funcionales-2/.

[15] «https://enmilocalfunciona.io/creacion-test-automatizado-con-cucumber-java-selenium-y-appium/,» [En línea].