



# Testing Básico.

11/06/2021

---

David Cebrián Cortés

IES Jacarandá

Brenes, (Sevilla)

# ÍNDICE

<b>ÍNDICE</b>	<b>1</b>
<b>INTRODUCCIÓN.</b>	<b>3</b>
¿Qué es testing?	3
<b>CLASIFICACIÓN DE LAS PRUEBAS.</b>	<b>4</b>
P. Manuales.	4
P. Automatizadas.	4
P. Unitarias	5
Pruebas de integración	5
P. funcionales	5
P. de regresión	6
P. de humo	6
P. de aceptación	6
P. de sistema	7
P. de compatibilidad	7
P. de usabilidad	7
P. de escalabilidad	8
P. de rendimiento	8
<b>HERRAMIENTAS</b>	<b>9</b>
<b>HERRAMIENTAS DE CODIGO ABIERTO.</b>	<b>9</b>
De GESTIÓN DE PRUEBAS	9
PARA P. FUNCIONALES	9
PARA P. DE CARGA Y RENDIMIENTO	9
<b>HERRAMIENTAS DE COMERCIALES.</b>	<b>9</b>
De GESTIÓN DE PRUEBAS	9
PARA P. FUNCIONALES	10
PARA P. DE CARGA Y RENDIMIENTO	10
<b>SELENIUM</b>	<b>10</b>
<b>SELENIUM IDE</b>	<b>11</b>
INSTALACIÓN	11
FUNCIONAMIENTO	12
PRUEBAS A REALIZAR	16
<b>SELENIUM WEB DRIVER</b>	<b>17</b>
INSTALACIÓN	17
FUNCIONAMIENTO	18
IDENTIFICACIÓN DE ELEMENTOS	20
MECANISMOS DE ESPERA	22
FORMULARIOS	23
PRUEBAS A REALIZAR	24

JASMINE & KARMA.	25
INSTALACIÓN Y FUNCIONAMIENTO	25
REFERENCIAS/BIBLIOGRAFÍA	28

# INTRODUCCIÓN.

## ¿Qué es testing?

Todo proceso de creación de software se compone de varias fases.

Una de esas fases es el **testing**, que a pesar de tener una gran importancia en todo desarrollo hay muchos casos en los que no se le da la necesaria o simplemente se omite esa parte. Es tan importante porque el **testing de software o software QA** es la parte dedicada a tener procesos de ejecución de un programa o aplicación y una metodología la cual se centra en encontrar errores en esa aplicación o software, o bien comprobar que todo funciona correctamente y puede ser desplegado en producción de una forma totalmente funcional.

Esta parte del desarrollo va paralela al propio desarrollo del software ya que a medida que se va construyendo un producto debemos hacer el testeo propio de las funcionalidades construidas para prevenir y corregir esos errores antes del lanzamiento de la aplicación.

A altos niveles se debe probar también que el software tiene una calidad mínima y que cumple unos requisitos previamente descritos por el cliente, aunque muchas veces errores propios de la comunicación entre el cliente y el desarrollador puede hacer que esos requisitos no se cumplan. Algunos de esos fallos pueden ser:

- Falta de detalles concretos.
- Malinterpretación de algún requisito.
- Problemas en la fase de diseño.
- Falta de alguna prueba necesaria.

Entregar un producto de buena calidad sería no tener ninguno de esos fallos y además comprobar mediante los test que la aplicación no tiene ningún error.

## CLASIFICACIÓN DE LAS PRUEBAS.

Existen muchos tipos de testing, o pruebas de software, que podemos usar para confirmar que nuestro software continúa funcionando correctamente tras introducir cambios nuevos sobre nuestro código fuente.

No todas las pruebas son iguales. Es por ello que vamos a ver en qué se diferencian las principales pruebas de software.

Vamos a aprender en qué consisten y en qué se diferencian tales tipos, como por ejemplo: unit testing, integration testing, functional testing, acceptance testing, y muchos más.

La clasificación de estas pruebas no es nada precisa y puede cambiar según la fuente de la información y un tipo también puede abarcar dentro de él otro de los que mencionamos, por lo que se ha decidido hacer la clasificación de la siguiente forma:

### P. Manuales.

Lo primero que debemos tener en cuenta es que existen pruebas manuales, las cuales las llevan a cabo personas interactuando ellas mismas con el sistema (aunque sea utilizando alguna herramienta para ello). Estas pruebas son más costosas ya que requieren que una persona las estén realizando cada vez que sea necesario y, además, está el factor del fallo humano a la hora de hacer las pruebas.

### P. Automatizadas.

Por otro lado tenemos las **pruebas automatizadas**, las cuales son ejecutadas por un script escrito previamente por un desarrollador. Éstas son más rápidas y confiables ya que no dependen del factor humano en su ejecución, pero la calidad de esas pruebas dependen de lo bien que el script está escrito. También son la clave para la integración continua ya que estos test se ejecutan cada vez que hay algún cambio para comprobar que las funcionalidades de la aplicación siguen funcionando correctamente.

## P. Unitarias

Estas pruebas consisten en probar las funciones o métodos de la aplicación de forma individual. Son muy específicas y por ellos son las de menor coste y las que se ejecutan más rápido en un servidor de integración continua.

- Estas pruebas verifican que el nombre de la función o método sea adecuado, que los nombres y tipos de los parámetros sean correctos, y que el tipo y valor de lo que se devuelve como resultado sea correcto.
- Dado que las pruebas unitarias no deben tener ningún tipo de dependencia, se suele reemplazar los llamados a APIs y servicios externos por funcionalidad que los imite. En muchos casos inclusive se suele reemplazar las consultas a bases de datos.
- Si no es factible aislar el uso de bases de datos de nuestras pruebas unitarias, será importante tener en cuenta el rendimiento y buscar optimizar nuestras consultas.
- Si nuestras pruebas unitarias son de larga duración, ralentizará significativamente el tiempo de despliegue y con ello los tiempos de desarrollo.

## Pruebas de integración

Las pruebas de integración verifican que los diferentes módulos y/o servicios usados por nuestra aplicación funcione en armonía cuando trabajan en conjunto.

Pueden:

- Probar la interacción con una o múltiples bases de datos.
  - Asegurar que los microservicios, (los cuales son como pequeños servicios o aplicaciones independientes que se comunican entre sí para formar una aplicación completa) operen como se espera.
- Son el siguiente paso de las unitarias y suelen ser más costosas de ejecutar.

## P. funcionales

Estas pruebas verifican el resultado de una acción, sin prestar atención a los estados intermedios del sistema mientras se lleva a cabo la ejecución.

Hay cierta confusión entre "integration tests" y "functional tests", ya que ambos requieren que múltiples componentes interactúen entre sí.

La diferencia es que una prueba de integración puede simplemente verificar que las consultas a una base de datos se ejecuten correctamente, mientras que una prueba funcional esperara un valor específico para un usuario.

### **P. de regresión**

Estas pruebas comprueban que las nuevas funcionalidades que hemos añadido a nuestro desarrollo no han afectado a otras anteriores ya desarrolladas y que ésta también funciona correctamente.

Un fallo en uno de estos test indica que la nueva funcionalidad ha afectado a una anterior o que se ha vuelto a producir un error anteriormente arreglado.

### **P. de humo**

Son pruebas que verifican la funcionalidad básica de una aplicación, deben ser rápidas de ejecutar y asegurar que las características principales de la aplicación funcione.

Ocurren a nivel general y deben estar pensadas para ejecutarlas a diario, de modo que si una función básica falla, poder arreglarlo de inmediato antes de desplegar nuevos cambios.

### **P. de aceptación**

Las pruebas de aceptación son pruebas formales, ejecutadas para verificar si un sistema satisface sus requerimientos de negocio.

Estas pruebas requieren que el software se encuentre en funcionamiento, y se centran en replicar el comportamiento de los usuarios, a fin de rechazar cambios si no se cumplen los objetivos. Estos objetivos pueden ir más allá de obtener una respuesta específica, y medir el rendimiento del sistema.

Son usualmente un conjunto de pruebas manuales que se realizan luego de que una fase de desarrollo ha finalizado por si algo no está correcto, poder volver a hacerlo rápidamente.

Comprueban que las características desarrolladas van acorde a los criterios de aceptación y especificaciones iniciales de los que partimos.

Suelen realizarse después de las pruebas unitarias o de integración, para evitar que se avance mucho con el proceso de prueba, y determinar a tiempo si se necesitan cambios significativos.

Es importante que los responsables definan esos criterios de aceptación de la forma más precisa posible y antes de empezar el proyecto para que este tipo de pruebas se lleven a cabo correctamente, y cualquier cambio o requerimiento que vaya surgiendo durante el desarrollo también deberá verse reflejado correctamente.

## P. de sistema

Estas pruebas no están destinadas a probar funcionalidades concretas dentro de nuestro desarrollo de la aplicación, sino que van más destinadas a probar operaciones o gestión del sistema como puede ser la escalabilidad, el rendimiento, la carga, la compatibilidad o la mantenibilidad.

### P. de compatibilidad

Comprueban que nuestra aplicación o sistema funciona en distintos entornos como pueden ser en distintos navegadores o en distintos sistemas operativos.

Estas pruebas se realizan porque al pasar de un sistema a otro puede haber **fallos estéticos** como el descuadre del texto o de la estructura fijada; y **fallos funcionales**, los cuáles pueden ser que una función como un botón funcione en un entorno y en otro no lo haga.

### P. de usabilidad

Este tipo de pruebas son realizadas para comprobar si la aplicación cumple el propósito para el que ha sido diseñada.

Consisten en seleccionar a un grupo de usuarios los cuáles tienen que llevar a cabo una acción en la aplicación y esa interacción es analizada por los desarrolladores y especialistas.

Las medidas que se toman en esas pruebas son: la exactitud con la que se han realizado las acciones, el tiempo que los usuarios las han tardado en llevarla a cabo, si recuerdan el uso de la aplicación tras un tiempo sin usar la aplicación o cómo se siente emocionalmente tras su uso.



## P. de escalabilidad

Estas pruebas nos permiten determinar, como su nombre indica, la escalabilidad de un sistema.

La escalabilidad de un sistema es la capacidad que tiene este mismo de soportar una carga cada vez mayor de datos y trabajo y seguir satisfaciendo las necesidades iniciales. Ejemplos de esa carga cada vez mayor pueden ser: cada vez más usuarios o cada vez más transacciones diarias

## P. de rendimiento

Las pruebas de rendimiento son aquellas pruebas que someten a un sistema a una carga de trabajo con el fin de medir su velocidad, fiabilidad y estabilidad en esas condiciones de trabajo.

Varios objetivos de estas pruebas son: **Encontrar cuellos de botella**(Encontrar dónde se encuentra el problema si falla al haber un exceso de carga de usuarios o si los tiempos de respuestas son muy altos), **localizar problemas de rendimiento en la aplicación**( si alguna funcionalidad tiene un tiempo muy alto de respuesta, ver porqué ocurre y cómo optimizarlo) o **verificar el cumplimiento de los Acuerdos del Nivel de Servicio**( Si se prevé que la aplicación va a tener una cantidad de usuarios determinada y con esa cantidad los tiempos de respuestas serán 3 segundos,se pueden ejecutar pruebas con esas condiciones y ver si se cumplen).

# HERRAMIENTAS

Existen muchas herramientas distintas para poder realizar las pruebas mencionadas anteriormente, en este documento vamos a citar las más conocidas y vamos a centrarnos y desarrollar algunas en específico que diremos más adelante

El control de la calidad de software lleva consigo aplicativos que permiten realizar pruebas autónomas y masivas permitiendo así la verificación desde el punto de vista estático y de caja blanca, es decir pruebas donde se analiza el software sin ejecutar el software mediante el código fuente del mismo. Algunas de estas herramientas son:

## HERRAMIENTAS DE CODIGO ABIERTO.

Estas herramientas están disponibles para todos el que quiera probarlas.

### De GESTIÓN DE PRUEBAS

- FitNesse.
- qaManager.
- Test Environment Toolkit.

### PARA P. FUNCIONALES

- **Selenium.**
- SoapUI.

### PARA P. DE CARGA Y RENDIMIENTO

- JMeter.
- FunkLoad.

## HERRAMIENTAS DE COMERCIALES.

Estas son herramientas para las cuales tendrás que pagar para poder usarlas.

### De GESTIÓN DE PRUEBAS

- ApTest Manager.
- Silk Central.

## PARA P. FUNCIONALES

- QuickTest Pro.
- Ranorex.

## PARA P. DE CARGA Y RENDIMIENTO

- Forecast.
- LoadStorm.

Una vez mencionadas algunas de las herramientas disponibles vamos a proceder a centrarnos en una de ellas: **SELENIUM**.

## SELENIUM

Selenium es un entorno de pruebas que se utiliza para comprobar si el software que se está desarrollando funciona correctamente. Esta herramienta permite: grabar, editar y depurar casos de pruebas que se pueden automatizar.

Lo interesante de Selenium es que puedes hacer pruebas automatizadas, (editando acciones o creándose desde cero) para poder lanzar en cualquier momento cuando lo necesites y así hacer también pruebas de regresión con ello.

Su objetivo principal es comprobar que un software funcione correctamente. Esto se consigue programando el movimiento que realizaría un usuario para llevar a cabo una acción determinada en una aplicación o página web, y comprobando posteriormente que esa acción se ha llevado a cabo correctamente de una forma u otra, es decir, emula y automatiza las acciones del usuario y comprueba que se han realizado correctamente.

A día de hoy, Selenium tiene un conjunto de herramientas de software y cada una muestra una perspectiva diferente. Muchos programadores deciden utilizar uno o dos a la vez para automatizar su proyecto, pero es mejor conocer todas las opciones y entender para qué sirve Selenium:

## SELENIUM IDE

Es una extensión de Firefox y Chrome que permite escribir test de Selenium con las interacciones del usuario y ejecutarlos directamente desde el navegador. Puedes indicarle rutinas de navegación para luego ejecutarlas una y otra vez y detectar así, de una manera sencilla, posibles errores. Todo esto sin necesidad de tener conocimientos de ningún lenguaje de scripting de prueba.

Tiene una interfaz bastante fácil de usar, pero tiene bastantes límites, por lo que es preferible combinarlo con otras herramientas de Selenium o utilizar otras solamente.

Este entorno logra desarrollar scripts automáticamente al crear una grabación y de esa forma se puede editar el script para reproducir esa grabación de la forma en que queremos, lo que se puede lograr es que se navegue por la aplicación de forma automática, ya sea realizando clicks, rellenando formularios, verificar la existencia de un elemento, etc.

Estos scripts son generados en *Sel*anese, un idioma de scripting para Selenium, pero se pueden exportar a otro tipo de idiomas.

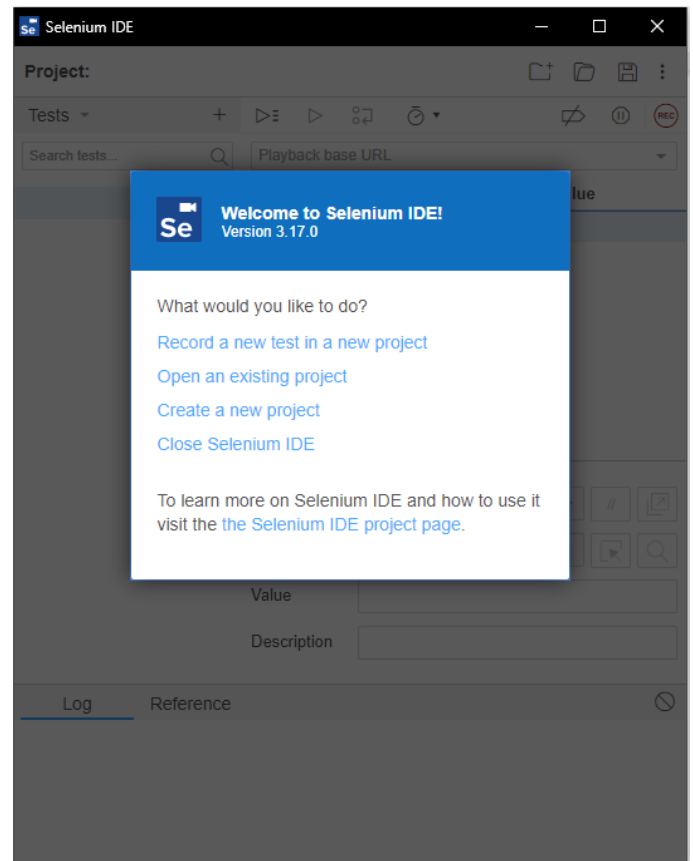
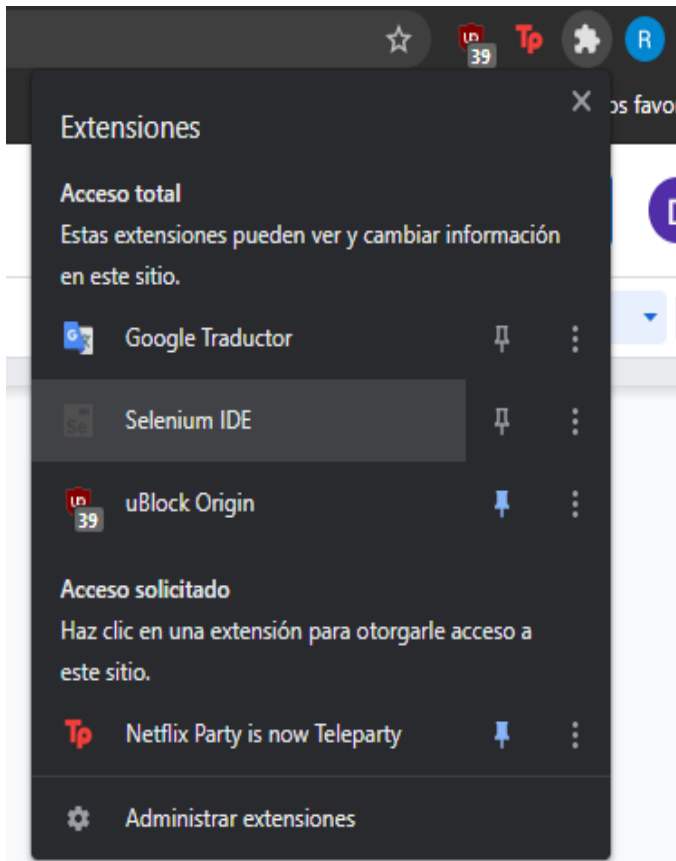
## INSTALACIÓN

Para la instalación de esta herramienta lo único que tenemos que hacer es irnos a la store de extensiones del navegador que queramos e instalarla. Nosotros lo haremos desde Chrome:

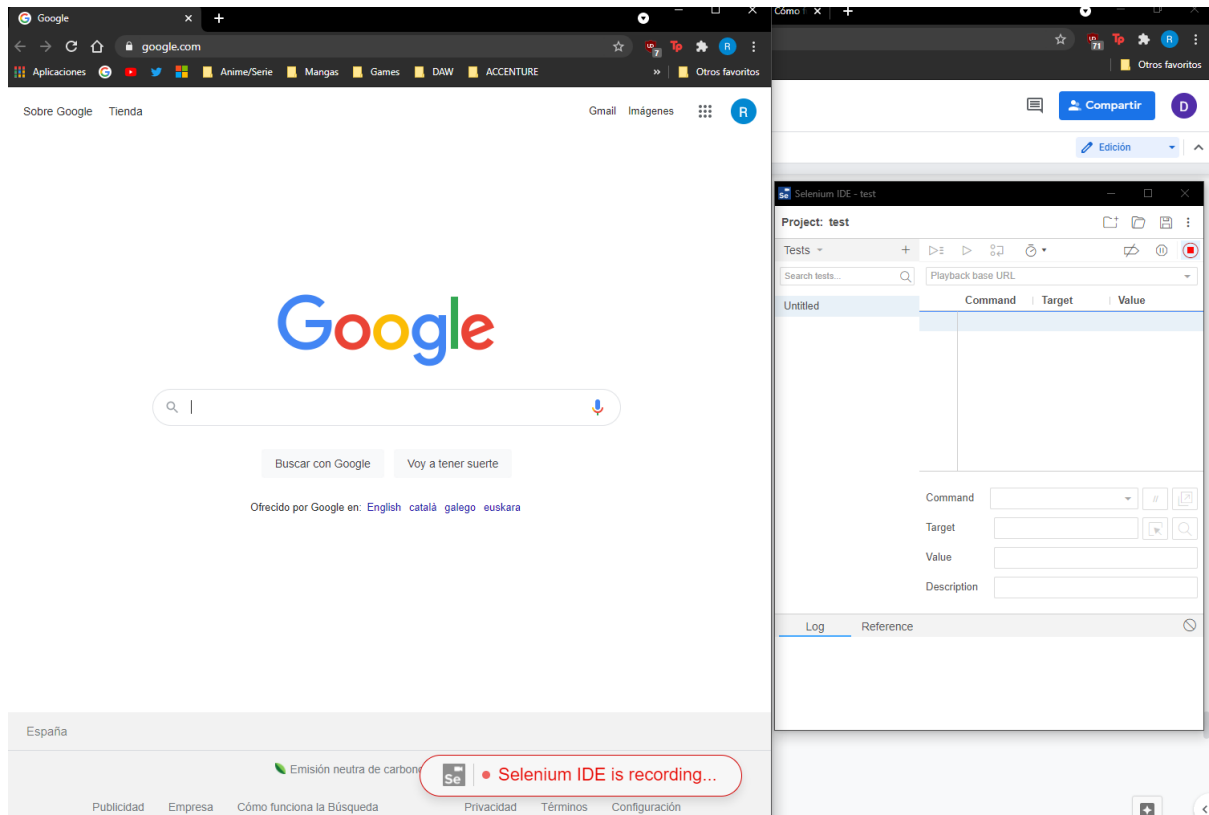
<https://chrome.google.com/webstore/detail/selenium-ide/mooikfkahbdckldjjndioac-kbalphokd/related>

## FUNCIONAMIENTO

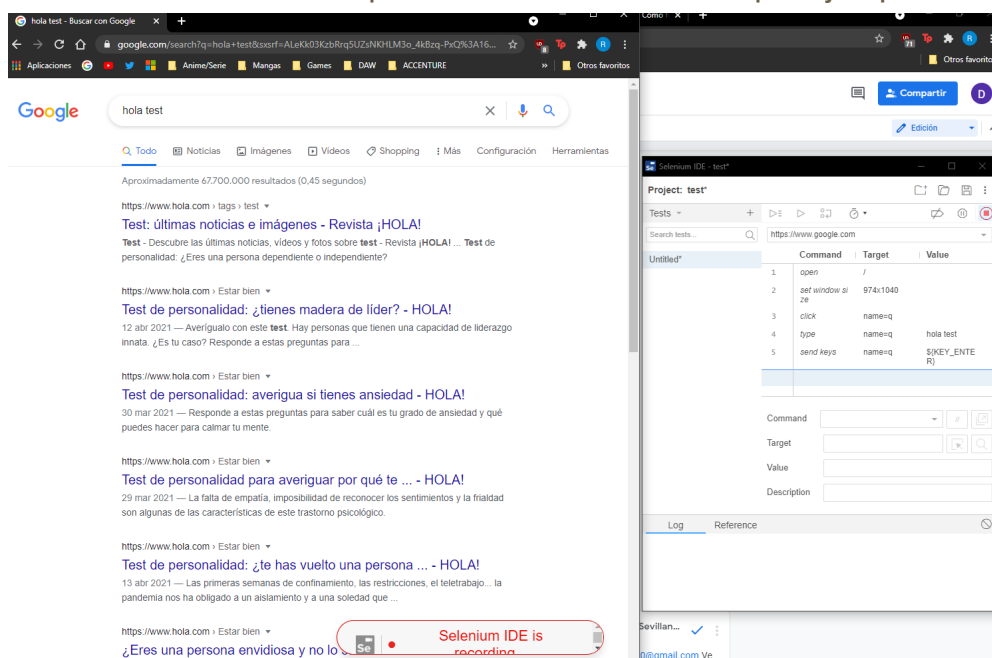
El funcionamiento de esta extensión o aplicación es bastante sencillo, lo único que tenemos que hacer es hacer click en las extensiones de nuestro navegador y clickar en SELENIUM IDE, esto abrirá la aplicación y nos preguntará que queremos hacer a continuación



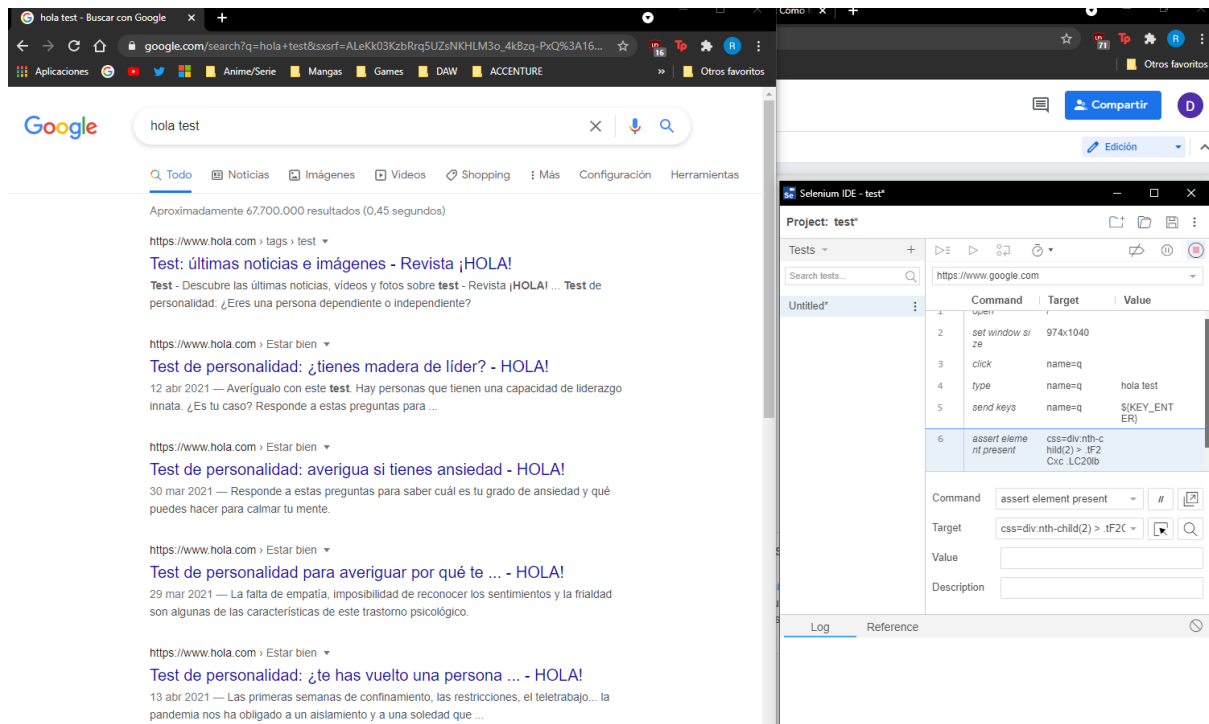
Una vez ahí hacemos click en new test in a new project y nos pedirá que ingresemos la URL desde la que queremos empezar a hacer las pruebas, para probarlo hemos introducido <https://google.com>, eso nos abre una ventana nueva con google.com abierto y la aplicación grabando automáticamente:



Ahora empezamos haciendo una búsqueda en google y veremos que se añaden comandos en la aplicación. Podemos añadir por ejemplo "hola test"



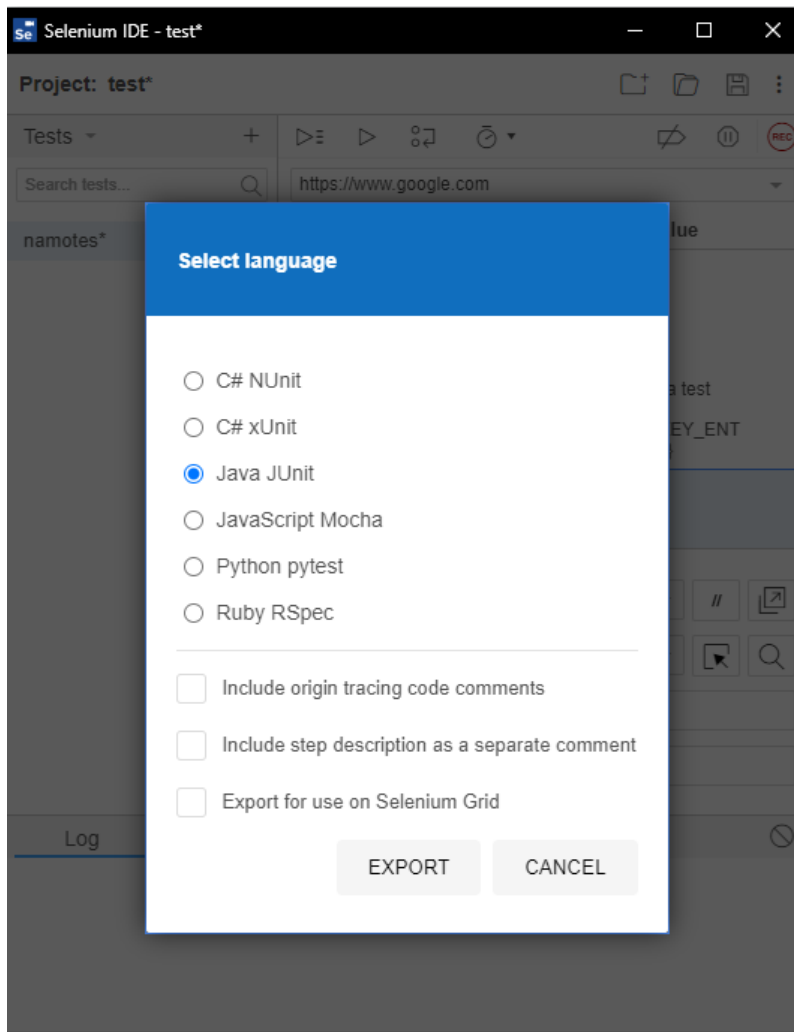
Y desde aquí para comprobar que hemos buscado correctamente hola test podemos comprobar que existe el primer link, esto lo hacemos clickando en **command** y en la lista seleccionamos **assert element present** y en target clickeamos en el icono de la ventana para poder seleccionar un elemento de la página en la que estamos. Ahora pulsamos en el icono de stop record arriba a la derecha:



Una vez pausemos nos pedirá que ingresemos un nombre y después podemos seleccionar el test que hemos creado en la lista de la izquierda y darle al play para ejecutarlo, y vemos que hace automáticamente los pasos que nosotros hemos hecho en la grabación. Podemos cambiar la velocidad a la que se ejecuta el test para darle tiempo a cargar las páginas antes de hacer las comprobaciones en el icono del reloj.



Una vez tenemos nuestro script lo podemos exportar a otro idioma para lanzarlo desde otro sitio haciendo click secundario en el test → export :



Éste es el funcionamiento más básico de Selenium IDE, a partir de ahí se pueden crear scripts y grabaciones de pruebas mucho más complejas y podemos lanzarlas cuando queramos.



## PRUEBAS A REALIZAR

Realizaremos las pruebas sobre la web de pc componentes

<https://www.pccomponentes.com/>, las pruebas van a ser grabadas con Streamlabs OBS ( <https://streamlabs.com/?l=es-ES> ) y editadas con OpenShot (<https://www.openshot.org/es/>).

Como demostración realizaremos unas pruebas en la página de **pccomponentes**.

- **Test Log IN**

La primera prueba que realizaremos será sobre el Login, comprobando una vez que nos logeemos que el elemento en el que aparece nuestra imagen de usuario exista.

Link:

[https://www.youtube.com/watch?v=qX4ZkIV3KGg&list=PLdTmqsgulk2X1zRnU94OY4seL3QN\\_DGbK&index=1](https://www.youtube.com/watch?v=qX4ZkIV3KGg&list=PLdTmqsgulk2X1zRnU94OY4seL3QN_DGbK&index=1)

- **Test Realizar pedido**

Otra prueba será comprobar que al clicar en un artículo nos lleva hacia la página de este y desde ahí si clickeamos en comprar nos lleva a la página de realizar el pedido comprobando que existe el botón de esa realización del pedido.

Link:

[https://www.youtube.com/watch?v=lZgM8NI8Fc8&list=PLdTmqsgulk2X1zRnU94OY4seL3QN\\_DGbK&index=2](https://www.youtube.com/watch?v=lZgM8NI8Fc8&list=PLdTmqsgulk2X1zRnU94OY4seL3QN_DGbK&index=2)

- **Test Carrito Compra**

Comprobaremos también que al hacer click sobre la opción de los artículos de añadir al carrito, éstos después existan dentro del carrito de la compra.

Link:

[https://www.youtube.com/watch?v=f1utvQ\\_1Nv4&list=PLdTmqsgulk2X1zRnU94OY4seL3QN\\_DGbK&index=1](https://www.youtube.com/watch?v=f1utvQ_1Nv4&list=PLdTmqsgulk2X1zRnU94OY4seL3QN_DGbK&index=1)

- **Test Búsqueda.**

Comprobaremos que al introducir en la barra de búsqueda algo, nos busque elementos relacionados con lo que hemos escrito. Escribiremos ordenador gaming y comprobaremos que nos ha buscado uno mediante su nombre.

Link:

[https://www.youtube.com/watch?v=UJmOntvd3PY&list=PLdTmqsgulk2X1zRnU94OY4seL3QN\\_DGbK&index=4](https://www.youtube.com/watch?v=UJmOntvd3PY&list=PLdTmqsgulk2X1zRnU94OY4seL3QN_DGbK&index=4)

## SELENIUM WEB DRIVER

Selenium Webdriver es una librería, disponible en distintos lenguajes de programación, que a través de un controlador específico de cada navegador (Chromedriver para Chrome, GeckoDriver para Firefox...) permite controlar una instancia del mismo.

Realiza acciones en el navegador y puede automatizarlas de igual forma que Selenium IDE pero permite mayor cantidad de posibilidades y no se utiliza una interfaz gráfica, si no que se programa en el idioma que vayamos a utilizar ya que está disponible para diferentes lenguajes de programación como son Java, Python, C#, JavaScript. Nuestro ejemplo será con Java mediante la realización de pruebas unitarias ejecutándolas con JUnit.

### INSTALACIÓN

#### Sin Maven

Para instalar las librerías de Selenium Webdriver en nuestro proyecto sin utilizar Maven ni el POM.xml lo que tenemos que hacer es:

- Descargarnos la librería de Selenium para el lenguaje que vayamos a utilizar (java en nuestro caso) desde su página oficial y las descomprimos: <https://www.selenium.dev/downloads/>
- Descargar el driver para nuestro navegador(chrome en nuestro caso): <https://chromedriver.storage.googleapis.com/index.html?path=90.0.4430.24/>
- Y en caso de que no lo tengamos aún, necesitamos descargar e instalar el JDK de Java desde: <https://www.oracle.com/latam/java/technologies/javase-downloads.html>

Ahora creamos un nuevo proyecto Java y creamos dos carpetas, una llamada *libs* en la que copiaremos las librerías descargadas anteriormente y otra llamada *drivers* donde introduciremos el driver de Chrome descargado. Ahora hacemos click secundario en el proyecto, Build path → Configure Build Path → Libraries > Add Jars → Seleccionamos las librerías que acabamos de copiar en nuestro proyecto y las añadimos.

#### Con Maven

Utilizando Maven y el POM.xml es mucho más sencillo ya que lo único que tendremos que hacer será añadir al POM la dependencia:

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

```
</dependency>
```

Y en el caso de que lo queramos para solo un navegador en concreto:

- **Chrome:**

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

- **Firefox:**

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

Y ahora añadimos el driver que hemos descargado a nuestro path con el siguiente comando:

- **Windows:**

```
setx /m path "%path%;C:\direccion_driver"
```

- **Linux:**

```
export PATH=$PATH:/opt/WebDriver/bin >> ~/.profile
```

Esto permitirá a Selenium localizar los binarios necesarios adicionales sin la necesidad de tener que incluir en el código de los tests la ruta exacta.

## FUNCIONAMIENTO

Una vez tengamos toda la configuración anterior ya podemos crear nuestra clase java dentro del proyecto, añadiendo el main.

Si no hemos añadido el driver al path tendremos que añadir una propiedad dentro de la clase, la cual es:

```
System.setProperty("webdriver.chrome.driver", "./drivers/chromedriver.exe");
```

Para evitarnos tener que añadir esa propiedad en cada clase se recomienda añadir al path el driver o drivers del navegador que vamos a utilizar.

Y para probar que todo funciona correctamente podemos crear una instancia del navegador esperar 5 segundos y cerrarla de esta forma:

```
1 Main_class.java 23
2
3
4
5
6 public class Main_class {
7
8     public static void main(String[] args) {
9
10         System.setProperty("webdriver.chrome.driver", "../drivers/chromedriver.exe");
11
12         WebDriver driver = new ChromeDriver();
13
14         try {
15             Thread.sleep(5000);
16         } catch (InterruptedException e) {
17             // TODO Auto-generated catch block
18             e.printStackTrace();
19         }
20
21         driver.quit();
22     }
23
24 }
```

Esta sencilla clase abre nuestro navegador según el driver que instanciamos, espera 5 segundos ( `thread.sleep(5000)` ) y se cierra( `driver.quit()` )

Siempre debemos introducir los pasos dentro de un try - catch - finally y dentro del finally cerrar el navegador y la sesión de selenium con: `driver.quit()`;

```
public static void main(String[] args) {

    //System.setProperty("webdriver.chrome.d

    WebDriver driver = new ChromeDriver();

    try {

        Thread.sleep(4000);

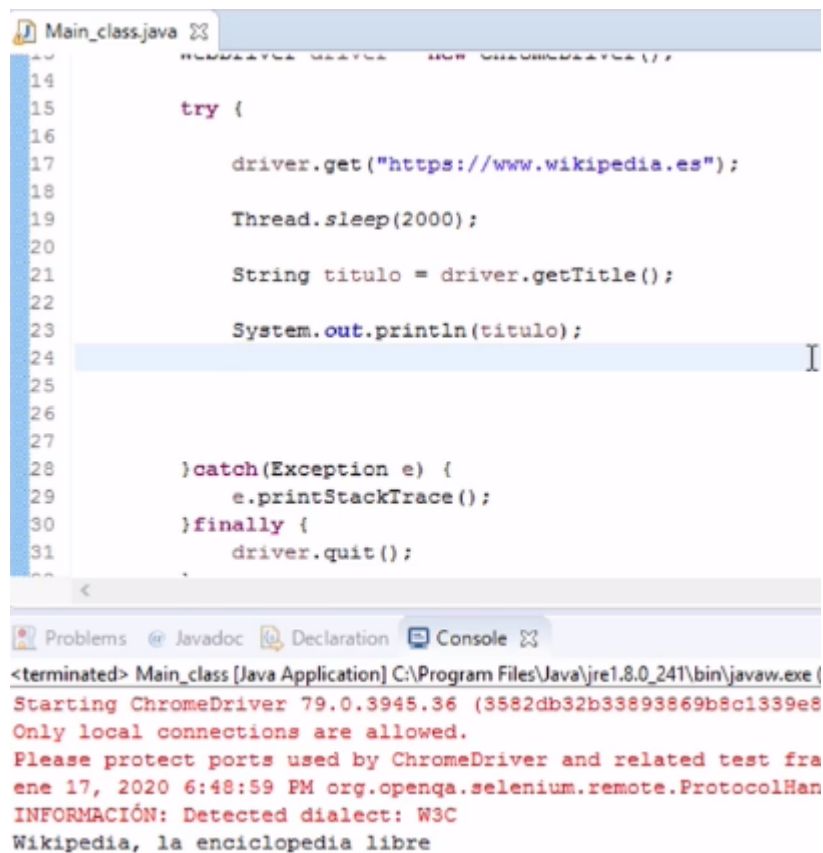
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        driver.quit();
    }
}
```

Y todos nuestros procesos irán dentro del try. Se deben ir añadiendo pequeños tiempos de espera( `Thread.sleep(time)` ) para darle tiempo al navegador a cargar las páginas o los elementos necesarios.

Por ahora solo hemos abierto el navegador, pero ahora debemos navegar hacia una página y mostrar su título por consola:

- Para navegar utilizamos: `driver.get("direccionpaginaweb")` ó `driver.navigate("direccionpaginaweb")`
- Y para seleccionar su título utilizamos: `driver.getTitle()`

- Y ya solo nos queda mostrar el string que nos devuelve por consola con `System.println()`



The screenshot shows an IDE with a Java file named `Main_class.java`. The code is as follows:

```
14 // ...
15
16 try {
17     driver.get("https://www.wikipedia.es");
18     Thread.sleep(2000);
19     String titulo = driver.getTitle();
20     System.out.println(titulo);
21 }
22
23
24
25
26
27
28 }catch(Exception e) {
29     e.printStackTrace();
30 }finally {
31     driver.quit();
32 }
```

Below the code editor, the console window is open, showing the following output:

```
<terminated> Main_class [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (
Starting ChromeDriver 79.0.3945.36 (3582db32b33893869b8c1339e8
Only local connections are allowed.
Please protect ports used by ChromeDriver and related test fra
ene 17, 2020 6:48:59 PM org.openqa.selenium.remote.ProtocolHan
INFORMACIÓN: Detected dialect: W3C
Wikipedia, la enciclopedia libre
```

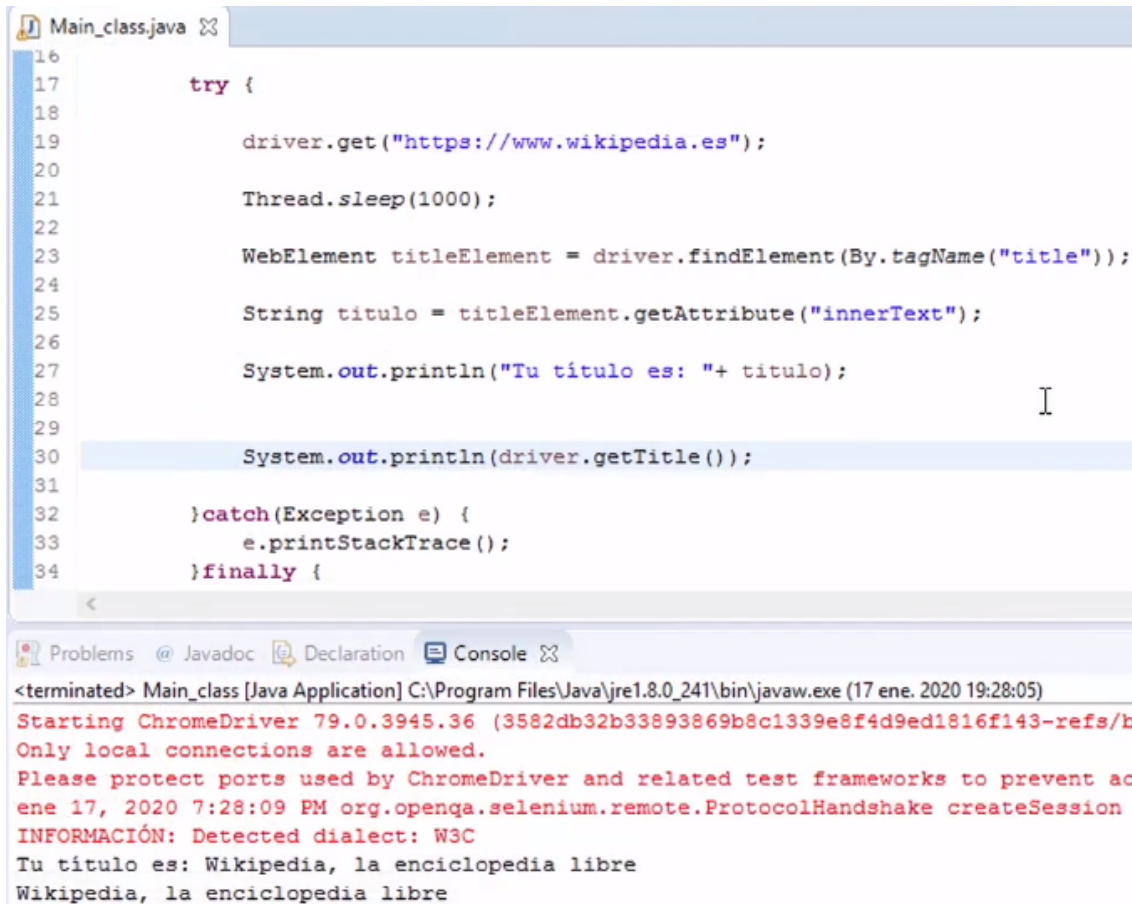
## IDENTIFICACIÓN DE ELEMENTOS

Para llevar a cabo las pruebas con selenium necesitaremos localizar y capturar los distintos elementos que nos hagan falta de la página o aplicación que estemos testeando, y para ello hay distintas formas de hacerlo.

Si queremos obtener un solo elemento utilizaremos `driver.findElement(By.atributo("hola"))`, donde "atributo" es el tipo de atributo o tag por el que queremos buscar el elemento, puede ser id, linkText, li, ul, o cualquier tipo de etiqueta de los elementos de la página. Nos devolverá el elemento como un objeto `WebElement`.

Si utilizamos este método en elementos que puedan existir más de uno, sólo nos devolverá el primero

Como ejemplo extraemos el título de la página a través de su tag y extrayendo el texto que contiene:



The screenshot shows an IDE with a Java file named 'Main\_class.java'. The code is as follows:

```
16
17     try {
18
19         driver.get("https://www.wikipedia.es");
20
21         Thread.sleep(1000);
22
23         WebElement titleElement = driver.findElement(By.tagName("title"));
24
25         String titulo = titleElement.getAttribute("innerText");
26
27         System.out.println("Tu título es: " + titulo);
28
29
30         System.out.println(driver.getTitle());
31
32     } catch (Exception e) {
33         e.printStackTrace();
34     } finally {
```

Below the code, the 'Console' tab is active, showing the following output:

```
<terminated> Main_class [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (17 ene. 2020 19:28:05)
Starting WebDriver 79.0.3945.36 (3582db32b33893869b8c1339e8f4d9ed1816f143-refs/b
Only local connections are allowed.
Please protect ports used by WebDriver and related test frameworks to prevent ac
ene 17, 2020 7:28:09 PM org.openqa.selenium.remote.ProtocolHandshake createSession
INFORMACIÓN: Detected dialect: W3C
Tu título es: Wikipedia, la enciclopedia libre
Wikipedia, la enciclopedia libre
```

Existen otras formas de seleccionar los elementos, las cuáles se pueden encontrar en la página oficial de selenium:

[https://www.selenium.dev/documentation/es/webdriver/locating\\_elements/](https://www.selenium.dev/documentation/es/webdriver/locating_elements/)

Pero dos más a mencionar ya que son importantes y tienen bastante precisión son:

- By.xpath: el cual lo sacamos de los elementos inspeccionando la página y haciendo click secundario en el html → copiar Xpath
- By.cssSelector: el cual es una forma de encontrar los elementos mediante el css.

Si queremos seleccionar más de un elemento al mismo tiempo debemos utilizar `driver.findElements()` de igual forma y esto nos devolverá una colección de `WebElements`.

## MECANISMOS DE ESPERA

Como las páginas web cargan elementos de forma asíncrona es necesario añadir mecanismos de espera a nuestras pruebas para que no haya errores al no cargar esos elementos a tiempo ( [NoSuchElementException](#) ).

Antes hemos mostrado una forma de espera que es parando la ejecución del hilo con `Thread.sleep()` pero esta no es la forma correcta de hacerlo ya que se puede perder mucho más tiempo al no saber cuánto tardará exactamente y tener que poner siempre algo más para asegurarnos. Por tanto los recomendados son:

- **Implicit Wait:** Es el más simple de todos y se aplica a todos los elementos que queramos buscar desde que se declara. Selenium va llamando al DOM hasta que le devuelve el objeto y si se acaba el tiempo especificado devuelve el error. Con esta forma no podemos especificar estados del elemento como por ejemplo esperar a que el botón esté habilitado para ser clicado. Se declara de la siguiente forma:

```
public static void main(String[] args) {  
    //CONFIGURACIÓN  
    System.setProperty("webdriver.chrome.driver", "C:\\proyecto_selenium\\chromedriver_win32\\chromedriver.exe");  
  
    //CREA LA INSTANCIA WEBDRIVER UTILIZANDO CHROME COMO NAVEGADOR  
    WebDriver driver = new ChromeDriver();  
  
    //CONFIGURAMOS LA INSTANCIA DE WEBDRIVER AGREGANDO UNA ESPERA IMPLÍCITA DE 10 SEGUNDOS  
    driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);  
}
```

- **Fluent Wait:** Se declara solo cuando vaya a ser usado y solo se aplica a los elementos que indiquemos, también nos permite ignorar errores y configurar los tiempos de esos intentos en los que Selenium consulta al DOM si ya se

```
Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)  
    .withTimeout(Duration.ofSeconds(15))  
    .pollingEvery(Duration.ofSeconds(3))  
    .ignoring(NoSuchElementException.class);  
  
WebElement boton = wait.until(new Function<WebDriver, WebElement>() {  
    public WebElement apply(WebDriver driver) {  
        return driver.findElement(By.id("boton1"));  
    }  
});
```

encuentra el elemento hasta que el tiempo se termine.



- **Explicit Wait:** Esta es la forma más correcta y más específica de realizar los métodos de espera, ya que nos permite aplicarlo a los elementos a los cuáles se lo indiquemos y además también nos permite indicar el estado en el que esperamos que el elemento se encuentre para que finalice la espera. Como el anterior también se declara cuando vaya a ser usado:

```
public static void main(String[] args) {  
  
    //CONFIGURACIÓN  
    System.setProperty("webdriver.chrome.driver", "C:\\proyecto_selenium\\chromedriver_win32\\chromedriver.exe");  
  
    //CREA LA INSTANCIA WEBDRIVER UTILIZANDO CHROME COMO NAVEGADOR  
    WebDriver driver = new ChromeDriver();  
  
    //CREAMOS LA INSTANCIA DE WEBDRIVERWAIT A LA QUE LLAMAREMOS PARA ESPERAR A QUE EL ELEMENTO CUMPLA LA CONDICIÓN QUE ESTABLECEMOS  
    WebDriverWait wait = new WebDriverWait(driver, 10);  
  
    //LLAMAMOS A LA INSTANCIA WAIT Y ESPECIFICAMOS QUE ESPERAMOS (QUE AL MENOS EN 10 SEGUNDOS) EL ELEMENTO ESTÉ DISPONIBLE PARA HACER CLICK  
    WebElement boton = wait.until(ExpectedConditions.elementToBeClickable(By.id("boton1")));  
}
```

## FORMULARIOS

Para rellenar formularios y enviarlos lo primero que debemos hacer es capturar el elemento que queramos rellenar ya sea un textarea, un checkbox o cualquier tipo de index, introducir el valor que queramos de la forma correcta en cada caso. Después cogemos el botón o el elemento con el que se envíe el formulario y lo clickeamos o lo utilizamos debidamente.

Los elementos más básicos y más utilizados, junto con su forma de rellenarlo en el formulario después de previamente haberlos extraído con las técnicas descritas en apartados anteriores son:

- TextBox → sendKeys('valor').
- ComboBox → selectByIndex(), selectByValue(), selectByVisibleText().
- CheckBox → click().
- RadioButton → click().
- Calendario → SendKey('DDMMAAAA' + Keys.TAB + 'hhmm').



## PRUEBAS A REALIZAR

Realizaremos las pruebas sobre la web de pc componentes <https://www.pccomponentes.com/>, las pruebas van a ser grabadas con Streamlabs OBS ( <https://streamlabs.com/?l=es-ES> ) y editadas con OpenShot (<https://www.openshot.org/es/>).

Como demostración realizaremos unas pruebas en la página de **pccomponentes**.

- **Crear Proyecto.**

Explicamos como añadir las dependencias y el driver a un proyecto Maven.

LINK:

[https://www.youtube.com/watch?v=Ge5iWw9gdP8&list=PLdTmqsgulk2U\\_hQ6zZinNaGvNq1hSTIJU](https://www.youtube.com/watch?v=Ge5iWw9gdP8&list=PLdTmqsgulk2U_hQ6zZinNaGvNq1hSTIJU)

- **Test Log IN**

La primera prueba que realizaremos será sobre el Login, comprobando una vez que nos logeemos que el elemento en el que aparece nuestra imagen de usuario exista.

Link:

[https://www.youtube.com/watch?v=XHYbjYbj-gc&list=PLdTmqsgulk2U\\_hQ6zZinNaGvNq1hSTIJU&index=2](https://www.youtube.com/watch?v=XHYbjYbj-gc&list=PLdTmqsgulk2U_hQ6zZinNaGvNq1hSTIJU&index=2)

- **Test Realizar pedido**

Otra prueba será comprobar que al clicar en un artículo nos lleva hacia la página de este y desde ahí si clickeamos en comprar nos lleva a la página de realizar el pedido comprobando que existe el botón de esa realización del pedido.

Link:

[https://www.youtube.com/watch?v=1fnyNdu6sKw&list=PLdTmqsgulk2U\\_hQ6zZinNaGvNq1hSTIJU&index=5](https://www.youtube.com/watch?v=1fnyNdu6sKw&list=PLdTmqsgulk2U_hQ6zZinNaGvNq1hSTIJU&index=5)

- **Test Carrito Compra**

Comprobaremos también que al hacer click sobre la opción de los artículos de añadir al carrito, éstos después existan dentro del carrito de la compra.

Link:

[https://www.youtube.com/watch?v=PM\\_xYH-hKLw&list=PLdTmqsgulk2U\\_hQ6zZinNaGvNq1hSTIJU&index=4](https://www.youtube.com/watch?v=PM_xYH-hKLw&list=PLdTmqsgulk2U_hQ6zZinNaGvNq1hSTIJU&index=4)

### - Test Búsqueda.

Comprobaremos que al introducir en la barra de búsqueda algo, nos busque elementos relacionados con lo que hemos escrito. Escribiremos ordenador gaming y comprobaremos que nos ha buscado uno mediante su nombre.

Link:

[https://www.youtube.com/watch?v=uWnUwajy6vE&list=PLdTmqsgulk2U\\_hQ6zZinNaGvNq1hSTIJU&index=3](https://www.youtube.com/watch?v=uWnUwajy6vE&list=PLdTmqsgulk2U_hQ6zZinNaGvNq1hSTIJU&index=3)

## JASMINE & KARMA.

Jasmine y Karma son las últimas herramientas que vamos a describir en este documento, estas sirven para testear las clases y componentes de nuestros proyectos desarrollados con Angular, y nos permiten realizar tanto test unitarios al probar funciones específicas como de regresión al probar distintas interacciones o llamadas que van pasando por distintos componentes o servicios.

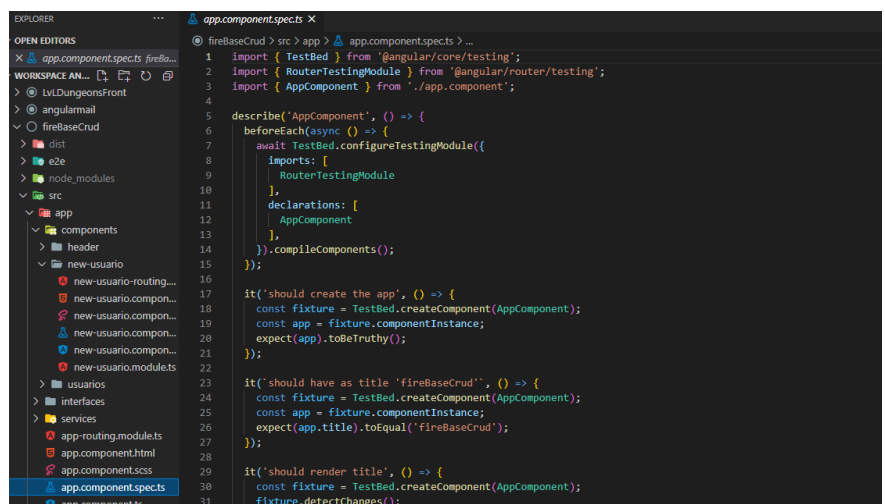
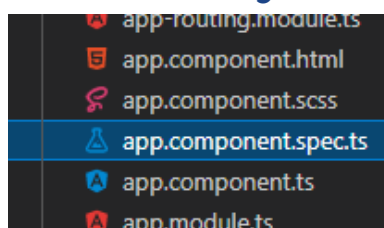
Los test en angular están pensados para llevar una metodología TDD( Test driven development) o BDD (Behavior driven development), en los que los test se realizan antes de desarrollar el código, para así llevar un control sobre lo que se está desarrollando y poder hacer cambios de una forma más efectiva al comprobar que nada se rompe cuando se hace.

Estas herramientas van de la mano ya que **Jasmine** es el framework que vamos a utilizar para crear nuestros tests y **Karma** es el programa que ejecuta esos test.

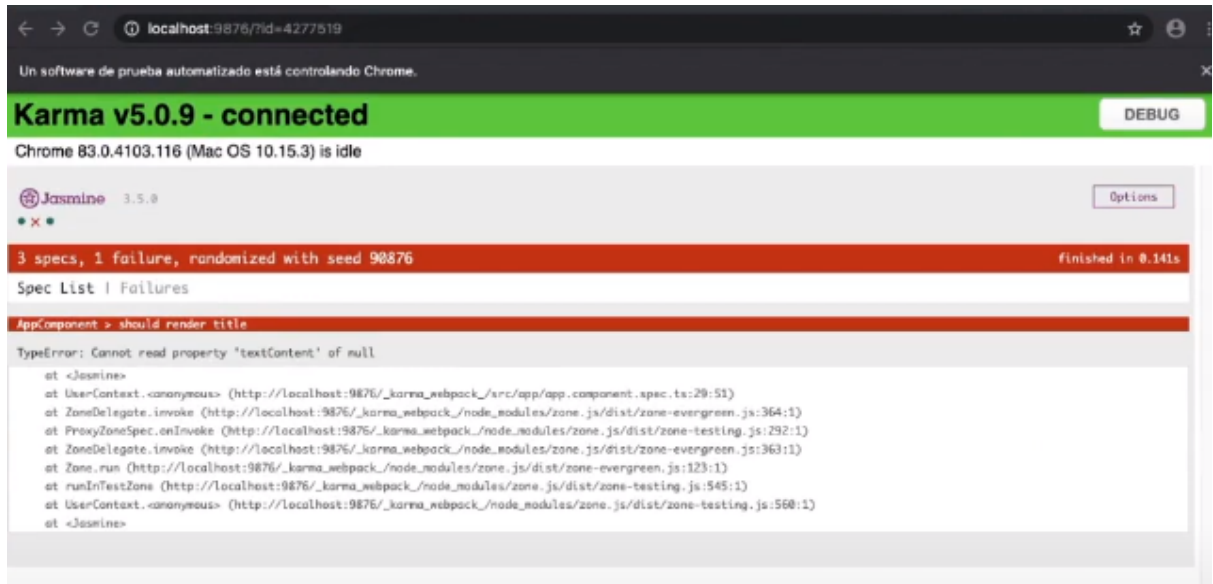
## INSTALACIÓN Y FUNCIONAMIENTO

Para instalar en nuestro proyecto Jasmine y Karma lo único que debemos hacer es crear nuestro proyecto Angular y al crearlo se nos crea por defecto un archivo.spec.ts, al igual que cuando creamos un componente también se nos crea ese archivo automáticamente.

Ese archivo es donde se hacen los test para posteriormente ser ejecutados con el comando **ng test**



Si no hemos tocado nada justo después de crear nuestro proyecto lanzamos este comando se nos abre un navegador con los resultados de los test automáticos que se crean, los cuáles deberán ser satisfactorios, y si ya hemos tocado algo como borrar el HTML autogenerado nos deberán fallar los test y mostrarnos una pantalla como esta:



Lo primero que nos encontramos es la importación de TestBed para poder ejecutar nuestros tests y el componente que vamos a testear.

TestBed es la primera y más importante de las utilidades para poder hacer pruebas en Angular. Crea un módulo angular de prueba que se configura con el método configureTestingModule para producir el entorno del módulo para la clase que desea probar.

```
import { ComponentFixture, TestBed } from '@angular/core/testing';

beforeEach(async () => {
  await TestBed.configureTestingModule({
    declarations: [ NewUsuarioComponent ]
  })
  .compileComponents();
});
```

Para realizar un test nos vamos al archivo .spec.ts del componente y para declarar una prueba o test tenemos la función **it()** dentro de un **describe()** donde describimos el componente que estamos haciendo las pruebas y mediante una función de flechas donde van todas las pruebas que vayamos a hacer con ese componente:

```
describe('NewUsuarioComponent', () => {
  let component: NewUsuarioComponent;
```

```
it(`should have as title 'fireBaseCrud`, () => {  
  const fixture = TestBed.createComponent(AppComponent);  
  const app = fixture.componentInstance;  
  expect(app.title).toEqual('fireBaseCrud');  
});
```

Dentro de la función debemos especificar es una descripción de nuestra prueba, y como segundo parámetro le añadimos lo que debe hacer nuestro test dentro de una función de flechas o “callback” y para hacer las comprobaciones o aserciones lo hacemos con **expect(lo que queremos comprobar).funcion()**. donde esa función puede ser:

- `toEqual(variable)` → Comprobamos que una cosa es igual a otra.
- `toBeTruthy/Falsy()` → Comprobamos un valor boolean.
- `toHaveBeenCalled()` → Compramos llamada a método.
- `toContain(var)` → Contiene variable.

Y muchas otras más.

Pero antes debemos hacer una instancia de nuestro componente y para que sea distinta para cada test lo hacemos dentro de un **beforeEach()**, al igual que la etiqueta **@before** en Java para que se ejecute antes de cada prueba:

```
beforeEach(() => {  
  fixture = TestBed.createComponent(NewUsuarioComponent);  
  component = fixture.componentInstance;  
  fixture.detectChanges();  
});
```

Aquí vemos como inicializamos nuestro componente. Podemos instanciarlo fuera del `beforeEach` y dentro lo inicializamos. Así ya podemos acceder a sus atributos y/o variables mediante `component.funcion` o `component.variable` para comprobar sus valores o resultados también podemos hacer lo mismo con los servicios.

En caso de que no queramos acceder directamente al servicio podemos crear un mock de él dentro del mismo archivo de test y acceder a sus variables y métodos como si fuese cualquier otro componente pero ahorrando el uso de otras dependencias internas del servicio como puede ser `LocalStorage` o llamadas a otras APIs externas:

```
class MockAuthService {  
  authenticated = false;  
  
  isAuthenticated() {  
    return this.authenticated;  
  }  
}
```

## REFERENCIAS/BIBLIOGRAFÍA

<https://www.hiberus.com/crecemos-contigo/testing-fase-de-testeo-de-software/>

<https://programacionymas.com/blog/tipos-de-testing-en-desarrollo-de-software>

[https://es.wikipedia.org/wiki/Pruebas\\_de\\_software](https://es.wikipedia.org/wiki/Pruebas_de_software)

[https://es.wikipedia.org/wiki/Pruebas\\_de\\_compatibilidad](https://es.wikipedia.org/wiki/Pruebas_de_compatibilidad)

[https://es.wikipedia.org/wiki/Prueba\\_de\\_usabilidad](https://es.wikipedia.org/wiki/Prueba_de_usabilidad)

<https://es.wikipedia.org/wiki/Selenium>

<https://www.adimedia.net/selenium-ide-un-herramienta-con-la-que-ahorras-tiempo-y-disgustos/>

<https://www.digital55.com/desarrollo-tecnologia/herramientas-testing-introduccion-selenium/>

<https://openwebinars.net/academia/aprende/selenium-principiantes/>  
[https://www.selenium.dev/documentation/en/getting\\_started/](https://www.selenium.dev/documentation/en/getting_started/)

<https://angular.io/guide/testing>

<https://medium.com/@jorgeucano/introducci%C3%B3n-al-testing-en-angular-da415ef8c47>

<https://codingpotions.com/angular-testing>