**Objetivo**

Utilizar [SelenumIDE](https://www.selenium.dev/selenium-ide/docs/en/introduction/getting-started) y VisualStudio code para ejecutar nuestras pruebas automáticas.

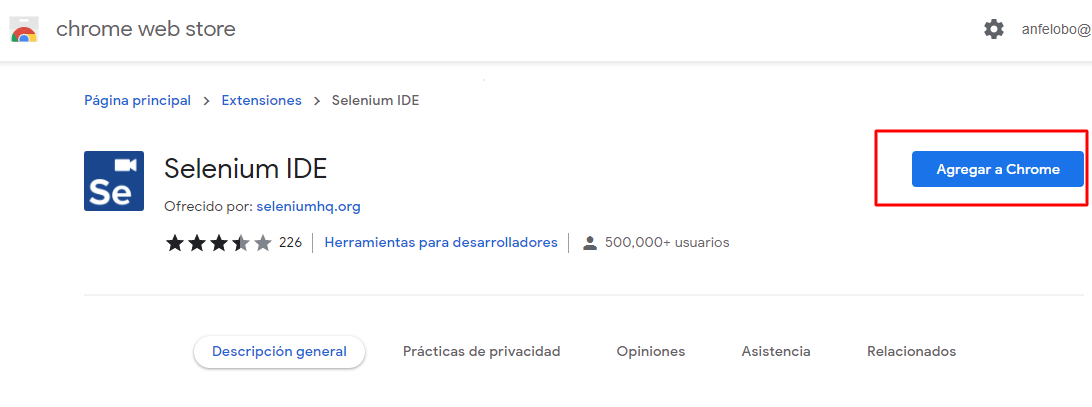
Para iniciar necesitamos, tener :

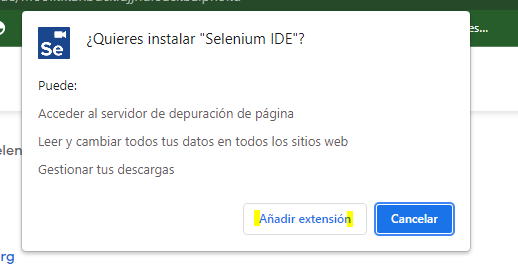
**SeleniumIDE :** Selenium IDE es el entorno de desarrollo del marco de pruebas, sobre el que se basa la extensión IDE para Chrome y Firefox. Se utiliza para grabar y reproducir pruebas.

**Webdriver:** Es la interfaz básica para simular las interacciones del usuario con un navegador, nuestro caso usaremos la versión **Chromewebdriver.**

Para iniciar, agregamos la extensión de SeleniumIDE, para ello buscamos en google o directamente desde el enlace:

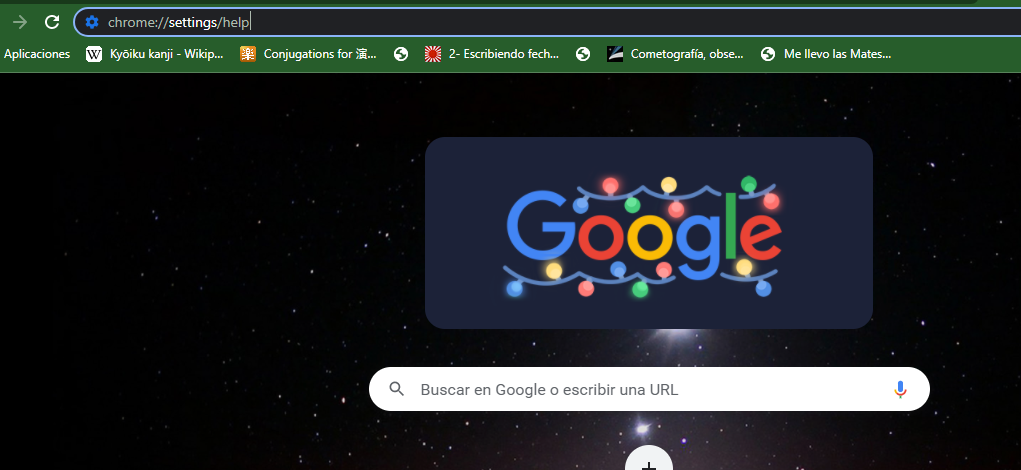
[SeleniumIDE](https://chrome.google.com/webstore/detail/selenium-ide/mooikfkahbdckldjjndioackbalphokd)

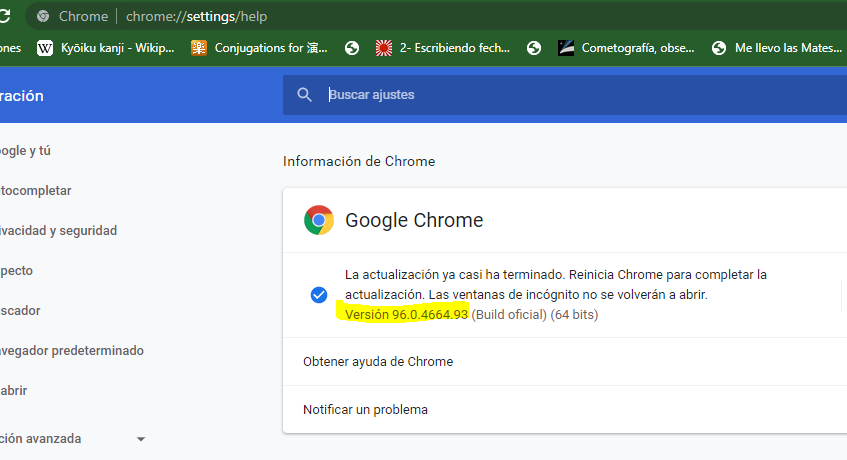




Para **Chromewebdriver,** primero debemos saber cuál es nuestra versión de chrome para descargar el que corresponde.

Digitamos en la barra de chrome: chrome://settings/help

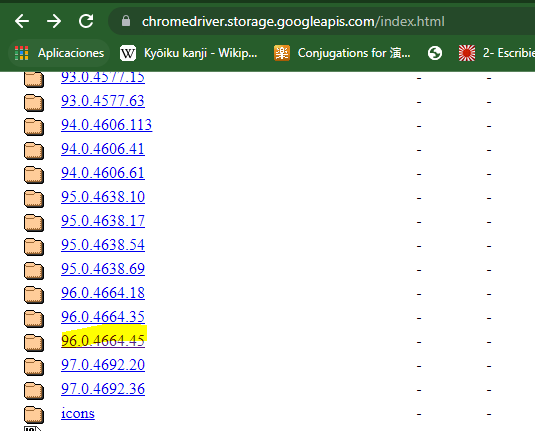


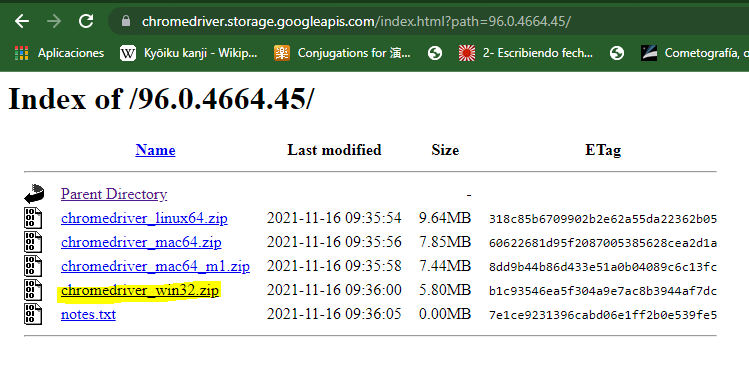


Ahora si buscamos nuestra versión en el siguiente enlace

[ChromeWebDriver](https://chromedriver.storage.googleapis.com/index.html)

Ingresamos a la carpeta y descargamos la que corresponda con nuestra versión.

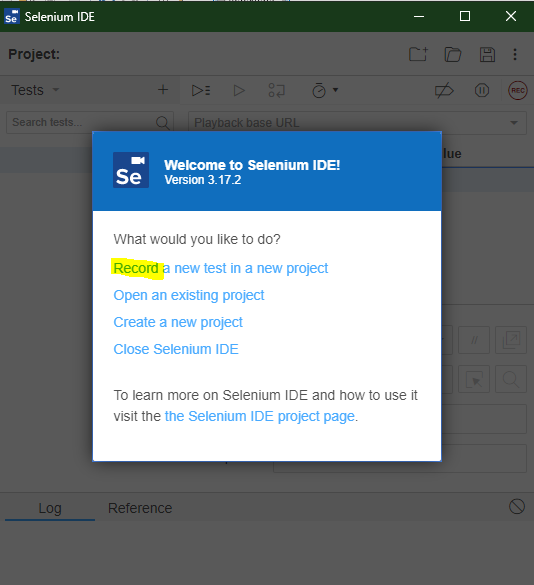




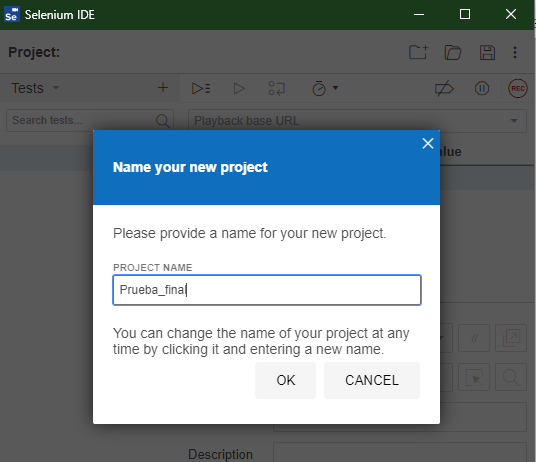
El driver se puede alojar en la carpeta de nuestro proyecto de pruebas o podemos setearlo en nuestras variables de entorno.

Para iniciar el **SeleniumIDE** vamos a la parte superior derecha de chrome y hacemos clic en la extensión

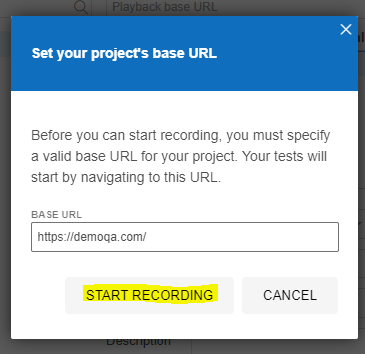
Seleccionamos record a new test…

****

Agregamos el nombre de nuestro proyecto de pruebas.

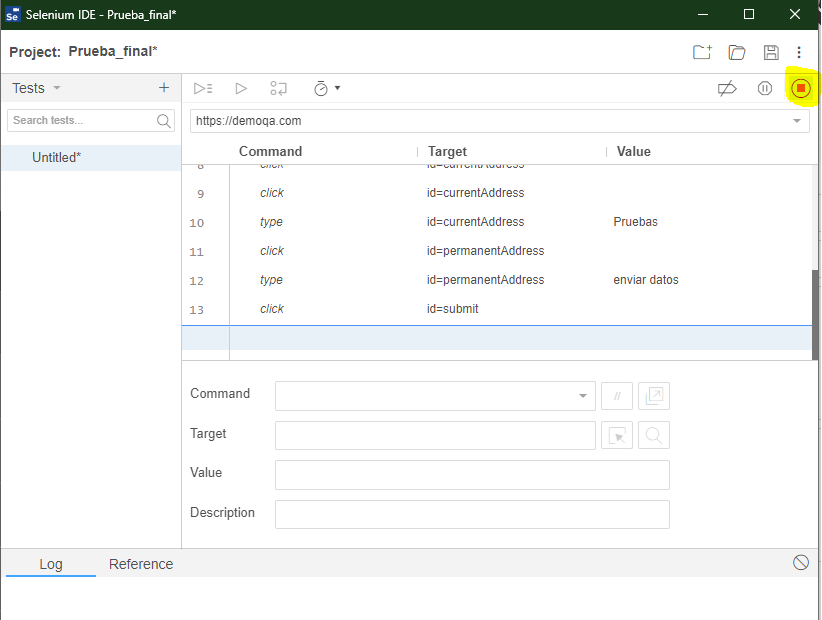


Indicamos cual es la URL donde se realizará la prueba e iniciamos la grabación

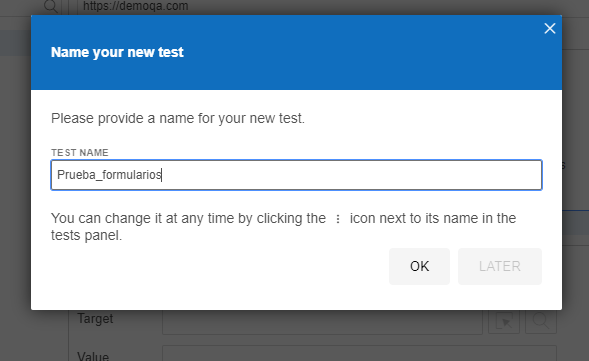


Se abrirá una nueva ventana y Selenium grabará todas nuestras acciones

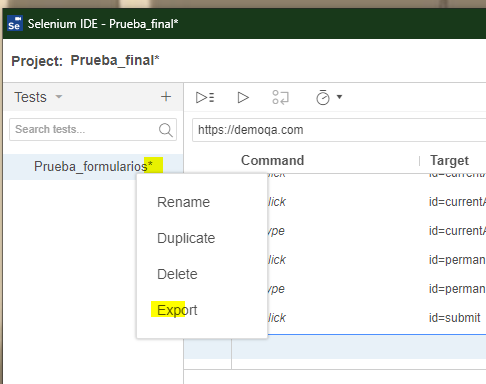
Una vez terminemos, hacemos clic en el botón : Stop recording (**Ctrl+u**)

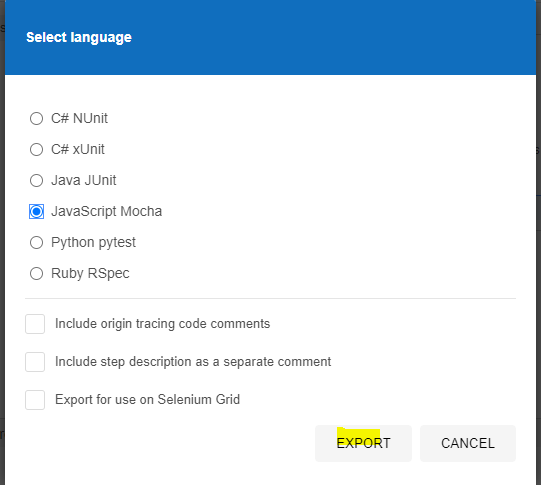


Agregamos el nombre de la prueba



y finalmente hacemos clic derecho sobre la prueba y exportamos

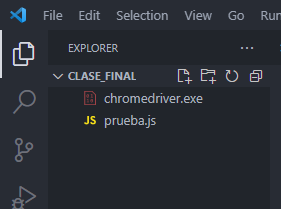




**En el visualStudio Code**

Creamos nuestro proyecto o ubicamos donde está creado si ya existe

Una vez ubicamos nuestro proyecto, debemos alojar allí el driver y el archivo JS que exportamos de SeleniumIDE.

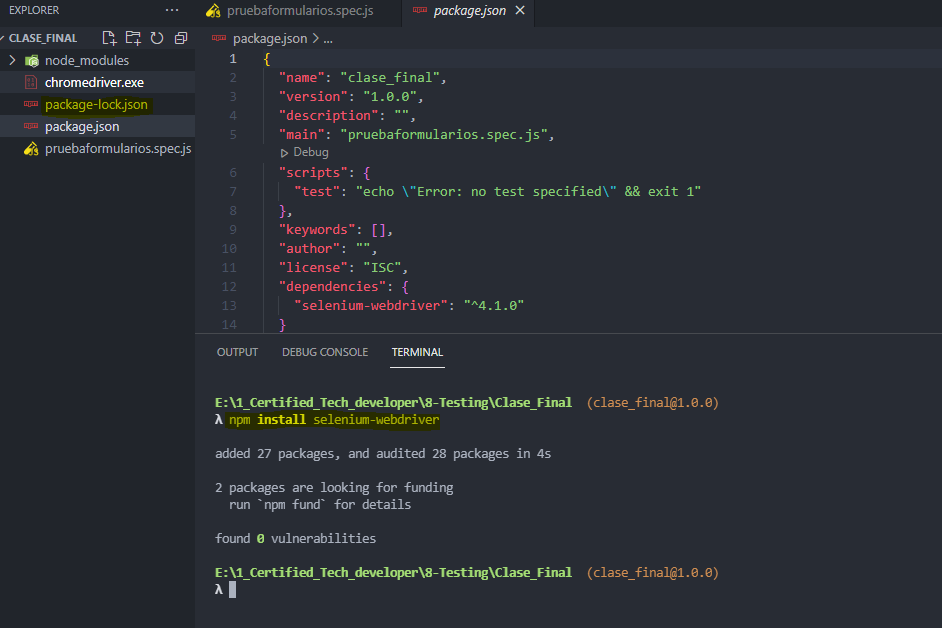


Abrimos la terminal (**Ctrl+ñ**) y preparamos nuestro entorno de pruebas ejecutando:

[npm init -y](https://docs.npmjs.com/cli/v8/commands/npm-init) iniciará las librerias de node.js que necesitamos para ejecutar nuestro javascript.

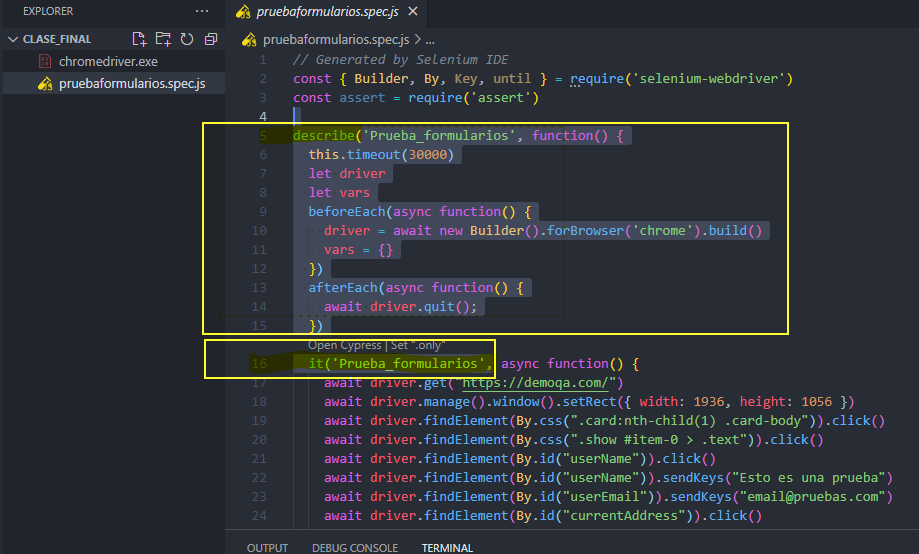
[npm install selenium-webdriver](https://www.npmjs.com/package/selenium-webdriver)

Al terminar se crearán los .json con las configuraciones (no es necesario editarlas)



En el visual, abrimos el archivo JS exportado desde seleniumIDE, este contiene la estructura mocha que no es compatible con selenium y debemos ajustarlo para que funcione correctamente.

Borramos desde describe hasta el nombre de la función



Debemos asegurarnos de borrar las llaves que ya no se necesitan y asignar el nombre la función asincrónica que vamos a utilizar, en este caso se ha nombrado “*nuestraFuncion*”



Ahora, para evitar errores en la ejecución, debemos agregar las siguientes líneas:

// Las capacidades son opciones que puede utilizar para personalizar y configurar una sesión de ChromeDriver. >

const chromeCapabilities=Capabilities.chrome();

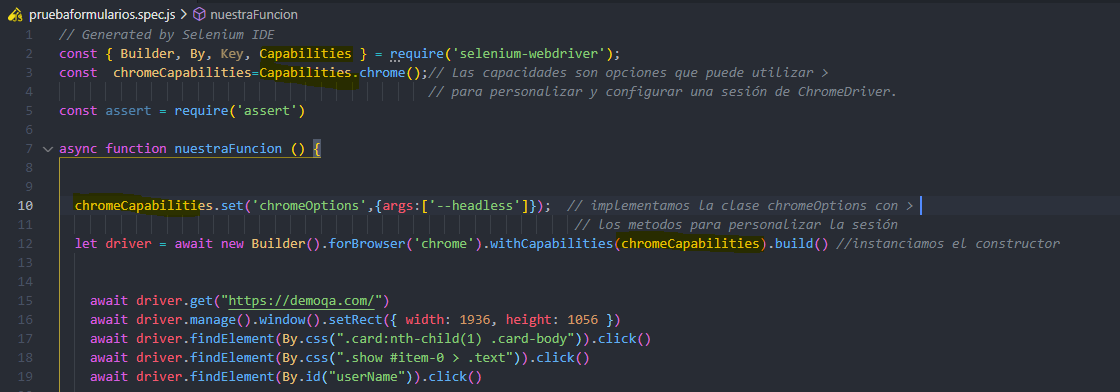
// implementamos la clase chromeOptions que contiene los métodos para personalizar la sesión. >

chromeCapabilities.set('chromeOptions',{args:cls[['**--headless**](https://peter.sh/experiments/chromium-command-line-switches/)']});

//instanciamos el constructor para que llame al driver de chrome >

let driver = await new Builder().forBrowser('chrome').withCapabilities(chromeCapabilities).build()

**Ejemplo:**



**Resultado final**

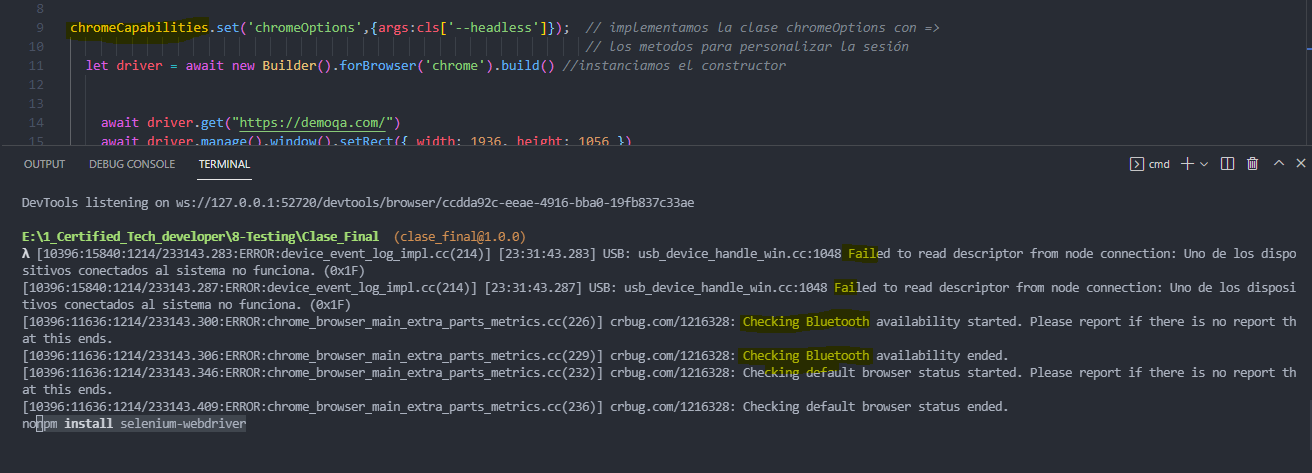


Cómo se observa en la parte final de la función hacemos el llamado correspondiente para que se pueda ejecutar la prueba, abrimos la terminal (**Ctrl+ñ**) y digitamos **node** **nombreDeNuestroArchivo.js** e iniciará la magia.

Se debe tener en cuenta que **no** se ejecuta npm test, esto solo se usa con jest, aquí estamos usando directamente al node.js para que ejecute nuestro javascript

**Notas**

* Si no utilizas [**chromeCapabilities**](https://developers.google.com/web/updates/2017/04/headless-chrome?hl=fr) nos va a generar el siguiente error:



Esto se da por que al ejecutar el driver trata de de utilizar la interfaz de usuario y se hace necesario pasar el siguiente parámetro **--headless**

* Estructura básica de las pruebas en selenium en visualstudio code:

const {Builder, By, Key, until} = require('selenium-webdriver');

async function example() {

let driver = await new Builder().forBrowser(chrome).build();

try {

await driver.get('http://www.google.com/ncr');

await driver.findElement(By.name('q')).sendKeys('webdriver', Key.RETURN);

await driver.wait(until.titleIs('webdriver - Google Search'), 1000);

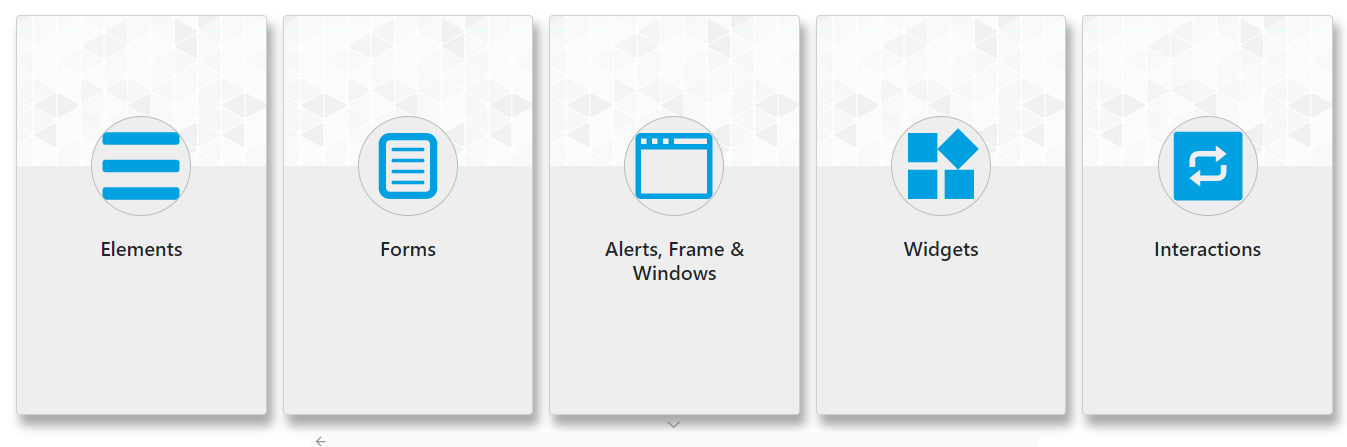
} finally {

await driver.quit();

}

} example();

* Para probar como funciona SeleniumIDE en diferentes componentes de una página web, podemos hacerlo desde <https://demoqa.com/>



* Links de referencia

<https://www.tutorialselenium.com/2019/09/02/usando-capabilities-y-chromeoptions-para-chromedriver/>

<https://sites.google.com/a/chromium.org/chromedriver/capabilities>

<https://www.npmjs.com/package/selenium-webdriver>

<https://peter.sh/experiments/chromium-command-line-switches/>

<https://developers.google.com/web/updates/2017/04/headless-chrome?hl=fr>

Canal donde explican sobre SeleniumWebdriver a un nivel más avanzado

<https://www.youtube.com/channel/UCVD0XAaYNHSb_YhaT9D-P3Q>