Índex

[**1.** **Proves Funcionals** 2](#_Toc163478201)

[**2.** **Requisits** 2](#_Toc163478202)

[**2.1.** **Instal·lar Docker** 2](#_Toc163478203)

[**2.2.** **Instal·lar JDK** 2](#_Toc163478204)

[**2.3.** **Instal·lar Maven** 2](#_Toc163478205)

[**2.4.** **Instal·lar IDE Visual Studio Code** 3](#_Toc163478206)

[**2.5.** **Accés a la intranet de git** 3](#_Toc163478207)

[**3.** **Descarregar el projecte** 3](#_Toc163478208)

[**4.** **Aixecar la arquitectura de serveis definida en el YAML anterior.** 5](#_Toc163478209)

[**5.** **Explicació de la plantilla/llibreria** 8](#_Toc163478210)

[**5.1.** **Arxius pertanyents de la llibreria** 9](#_Toc163478211)

[**5.1.1.** **ConfigParameters** 9](#_Toc163478212)

[**5.1.2.** **BaseTest.java** 10](#_Toc163478213)

[**5.1.3.** **Utils.java** 10](#_Toc163478214)

[**5.1.4.** **BrowserOptions.java** 11](#_Toc163478215)

[**5.1.5.** **ExtentManager.java** 11](#_Toc163478216)

[**5.1.6.** **ResultSender.java i ExecutionListener.java** 11](#_Toc163478217)

[**5.2.** **Compilar i executar la plantilla** 12](#_Toc163478218)

[**6.** **Crear casos de proves** 14](#_Toc163478219)

[**6.1.** **Definir Page Objet Models** 15](#_Toc163478220)

[**6.2.** **Definir Test** 23](#_Toc163478221)

[**6.3.** **Definir test.xml** 27](#_Toc163478222)

[**7.** **Executar les proves des de Visual Studio Code** 27](#_Toc163478223)

[**8.** **Anàlisi de resultats en el report HTML.** 30](#_Toc163478224)

[**8.1.** **Anàlisi de l'error** 33](#_Toc163478225)

[**8.1.1.** **Solucionar l'error** 35](#_Toc163478226)

# **Proves Funcionals**

Les proves funcionals d'aplicacions a través de Selenium són un enfocament crucial en el desenvolupament de programari, on Selenium, una eina d'automatització de proves, s'utilitza per verificar el correcte funcionament d’aplicacions web. Aquestes proves es centren en validar la funcionalitat de l'aplicació des de la perspectiva de l'usuari, simulant interaccions reals de l'usuari, com fer clic en botons, omplir formularis i navegar per diferents pàgines.

Selenium permet crear scripts que emulen accions humanes, la qual cosa garanteix que l'aplicació compleixi amb els requisits funcionals i que es mantingui estable davant possibles canvis o actualitzacions.

 En el següent manual, podeu executar proves funcionals de Selenium, a través de Selenium Grid, una eina que permet orquestrar i escalar els navegadors.

# **Requisits**

## **Instal·lar Docker**

Assegureu-vos que teniu Docker y Docker Compose instal·lat al vostre sistema. Podeu descarregar-lo des del lloc web oficial de Docker o utilitzar un gestor de paquets si està disponible per al vostre sistema operatiu.

Link: [Get Docker](https://docs.docker.com/get-docker/)

Link: [Get Docker Compose](https://docs.docker.com/compose/install/)

## **Instal·lar JDK**

1. Descàrrega i instal·lació de [Java Development Kit (JDK).](https://www.oracle.com/java/technologies/downloads/)
2. Establir la variable d'entorn *JAVA\_HOME: C:\Program Files\Java\jdk1.8.0\_301*
3. Actualitzar la variable d'entorn *PATH: %JAVA\_HOME%\bin*

## **Instal·lar Maven**

1. Descàrrega i instal·lació de [Maven](https://maven.apache.org/download.cgi).
2. Establir la variable d'entorn *MAVEN\_HOME: C:\Program Files\Maven\apache-maven-3.8.8\bin*
3. Actualitzar la variable d'entorn *PATH: %MAVEN\_HOME%\bin*

## **Instal·lar IDE Visual Studio Code**

Per a la realització d'estàs proves és necessari tenir instal·lat una eina d'entorn de desenvolupament integrat.

En aquest exemple, s'utilitzarà el IDE [Visual Studio Code](https://code.visualstudio.com/download).

## **Accés a la intranet de git**

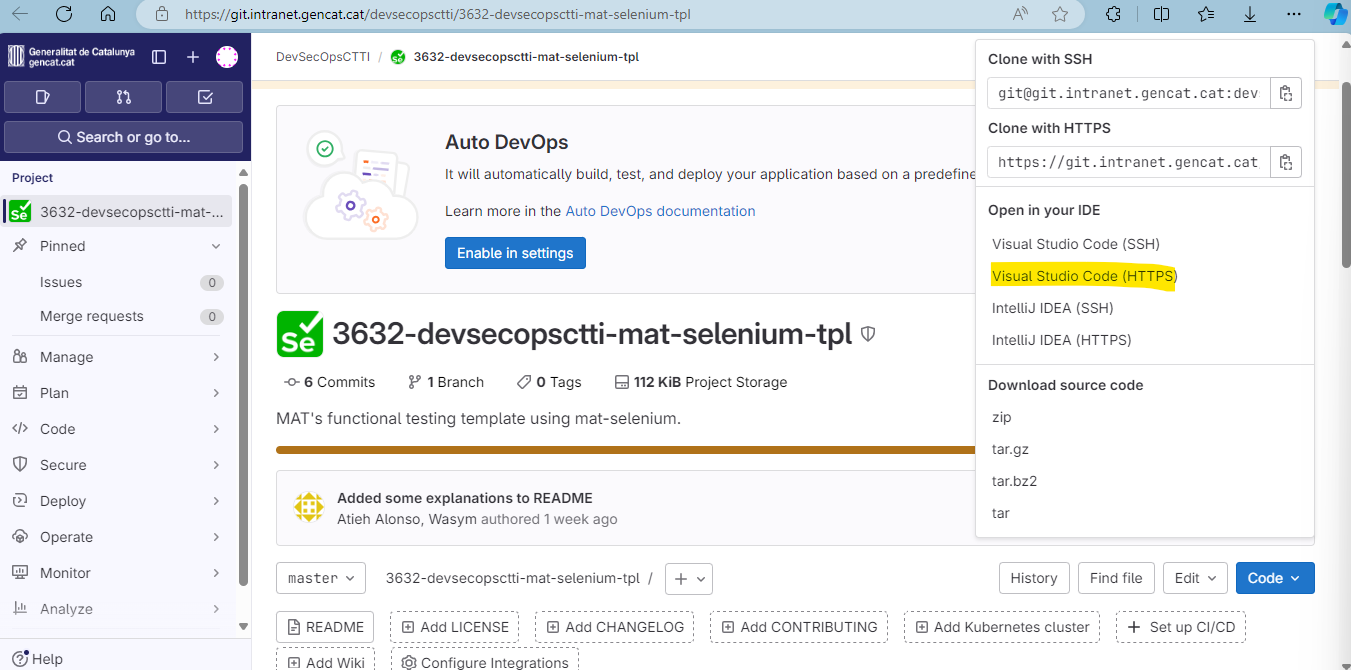
Comprovar que es té accés a [git.intranet.gencat.cat](https://git.intranet.gencat.cat/devsecopsctti/3632-devsecopsctti-mat-selenium-tpl) per a poder descarregar la plantilla del projecte.

# **Descarregar el projecte**

Accedir al repositori per a descarregar la plantilla del projecte.

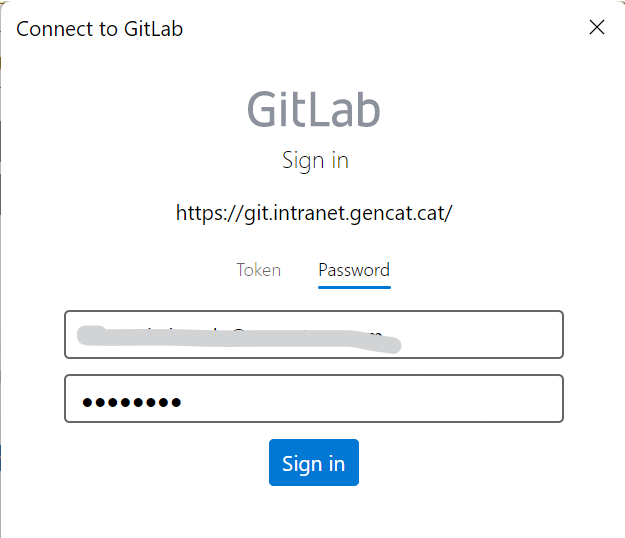
Aquest projecte conté els següents arxius:

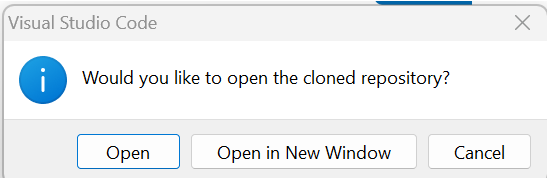
|  |  |
| --- | --- |
| **ELEMENT** | **DESCRIPCIÓ** |
| **config.properties** | Aquest arxiu conté els camps de configuració que es necessitin per a l'execució del projecte. |
| **src/test** | La carpeta emmagatzemarà tots els scripts, artefactes i recursos que es necessiten per a executar les proves, comprèn una jerarquia de carpeta entre les quals es troben main i test. |
| **docker-compose.yaml** | Fitxer d'execució del docker per a aixecar Selenium Grid. |
| **pom.xml** | És un arxiu XML del model d'objectes del projecte que conté informació sobre els detalls de configuració que Maven necessita. |
| **README** | Aquest arxiu conté informació sobre el projecte i com utilitzar-lo. És una forma de documentació de programari, usualment en un arxiu de text pla en format TXT, o MD. |



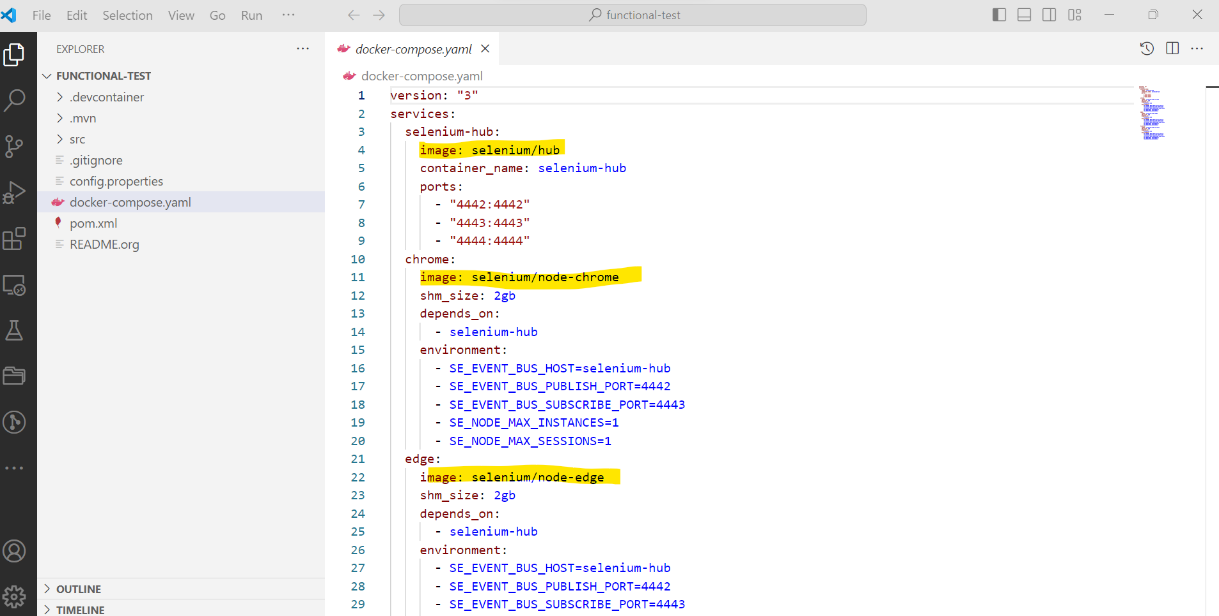
Obrir el projecte en el IDE De Visual Studio Code (HTTPS).

Connectar-se a GitLab amb les mateixes credencials i obre el projecte.





L'arxiu de *docker-compose.yaml*, es compon de la informació necessària per a executar Selenium Grid amb Docker.



Hi ha dos paràmetres que a tenir en compte que poden ser modificats pel tester, en funció de la concurrència que necessiti:

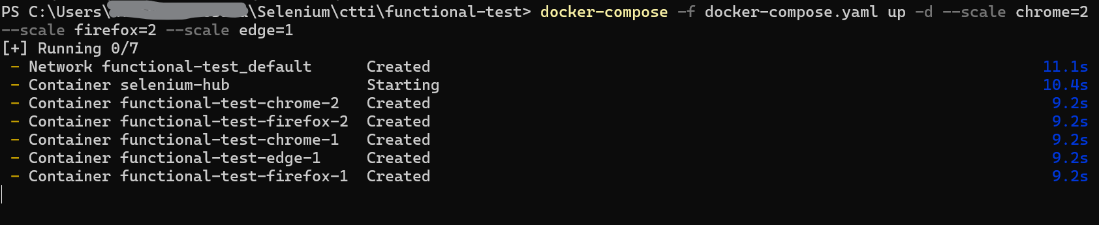
* SE\_NODE\_MAX\_INSTANCES=X
* SE\_NODE\_MAX\_SESSIONS=X

# **Aixecar la arquitectura de serveis definida en el YAML anterior.**

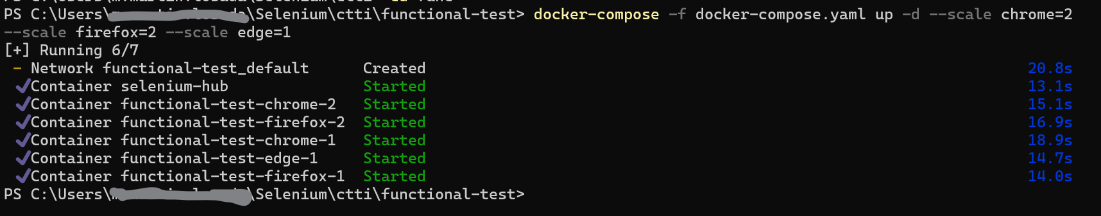
Obrir la consola (per exemple, Powershell) per a aixecar la arquitecutura definida en el docker-compose.yaml i accedir a Selenium Grid.

Accedir al directori on es trobi el projecte i executar la següent instrucció (els valors de cada navegador indiquen el nombre de nodes (contenidors) per cada navegador):

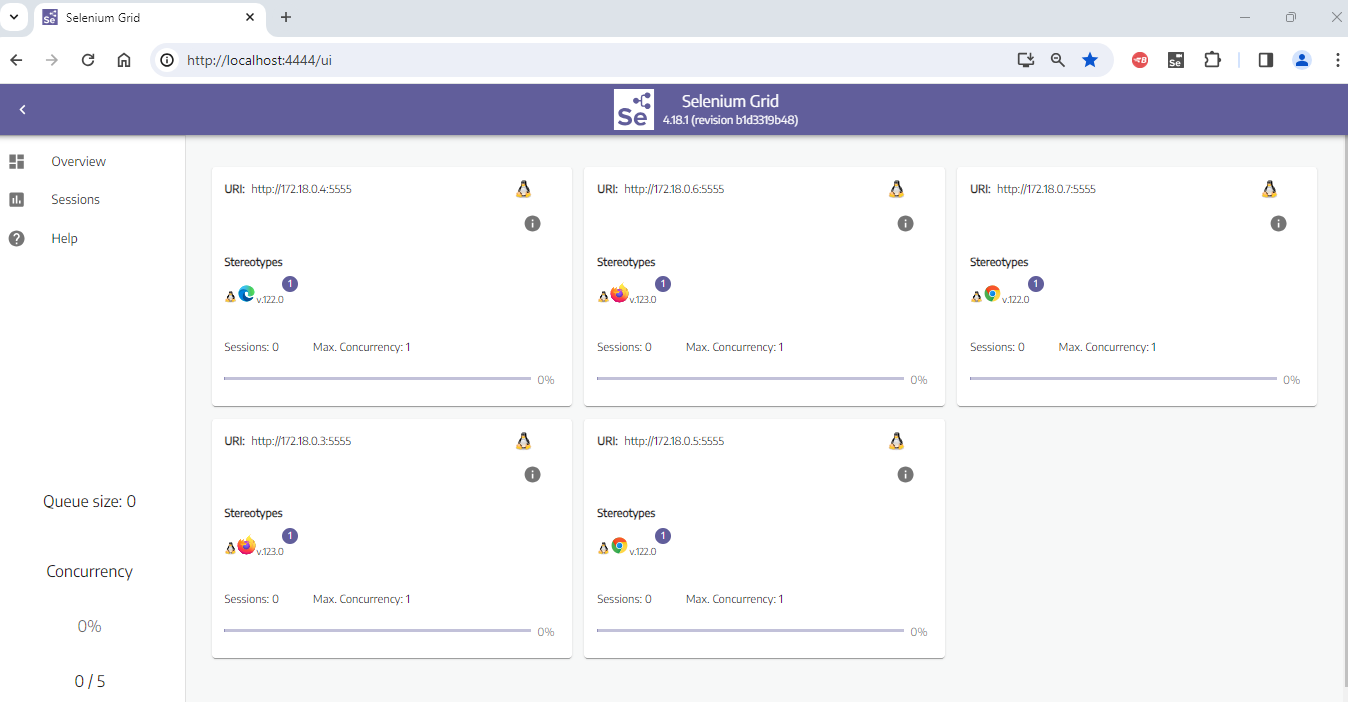
*docker-compose -f docker-compose.yaml up -d --scale chrome=2 --scale firefox=2 --scale edge=1.*



Esperar que tots els contenidors estiguin aixecats.

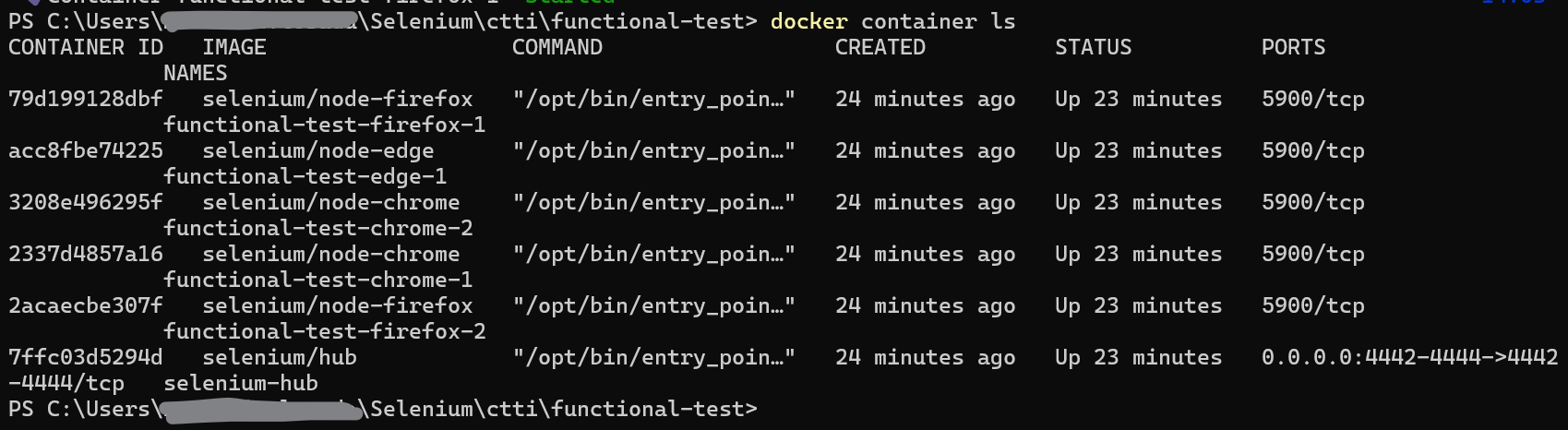


Accedir a la WebUI de Selenium Grid: <http://localhost:4444/ui>



També es pot veure per consola els contenidors aixecats:

*docker container ls*

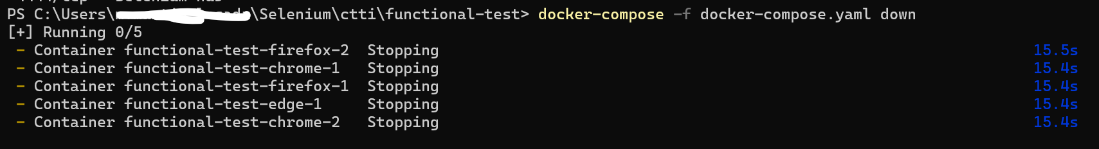


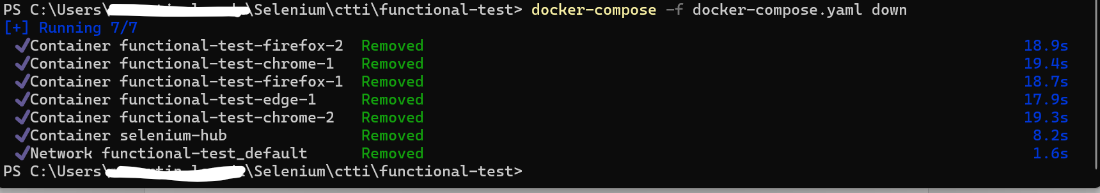
Una vegada que tots els contenidors estiguin aixecats i que tots els nodes estiguin disponibles en la url de Selenium Grid, ja es pot començar a executar les proves.

*Recordatori:*

Una vegada que hagis termiando d'executar les proves, és recomanable **eliminar els serveis prèviament aixecats** per a no consumir recursos.

*docker-compose -f docker-compose.yaml down*





# **Explicació de la plantilla/llibreria**

La plantilla descarregada en l'apartat 3 inclou la llibreria del MAT que inclou classes comunes que permeten la correcta execució de les proves. Aquestes classes inclouen mètodes com aixecar el Drive de Selenium, obrir el navegador, realitzar captures, monitorar els resultats en InfluxDB i Grafana, etc.

|  |  |
| --- | --- |
| **Arxius visibles en la plantilla** | **Arxius pertanyents de la llibreria** |
| * config.properties. * src/test.   + Pages > QualityGenCatPage.java   + QualityGenCatTest.java * docker-compose.yaml. * pom.xml * README: aquí es pot trobar [l'explicació de la plantilla](https://git.intranet.gencat.cat/devsecopsctti/3632-devsecopsctti-mat-selenium-tpl/-/blob/master/README.org?ref_type=heads). | * src/main/java:   + BaseTest.java   + BrowserOptions.java   + ConfigParameters.java   + ExecutionListener.java   + ExtentManager.java   + ResultSender.java   + Utils.java |

Els arxius pertanyents a la llibreria no estan visibles en la plantilla descarregada, ja que aquestes són classes comunes i no necessiten ser modificades pel tester.

Dins de la carpeta **src/test** es troba la carpeta Java que al seu torn conté dos elements:

* Pages > QualityGenCatPage.java: aquesta classe java és en la qual es defineixen els Pages Object Model (explicat en apartat 6.1)
* QualityGenCatTest.java: aquí es defineixen els passos del test que s'executaran. Aquest test estén de la classe **BaseTest.java** i utilitza mètodes definits en la classe **Utils.java**. Totes dues pertanyents a la llibreria.

## **Arxius pertanyents de la llibreria**

### **ConfigParameters**

Aquesta classe analitza i guarda els paràmetres de configuració (la url de l'aplicació, el mantenidor, les dades de InfluxDB...).

Aquestes són les variables d'entorn disponibles:

|  |
| --- |
| Protected static String app=System.getenv("MAT\_TF\_APP"); |
| Protected static String app\_url=System.getenv("MAT\_TF\_APP\_URL"); |
| Protected static String maintainer=System.getenv("MAT\_TF\_MAINTAINER"); |
| protected static String ambit = System.getenv("MAT\_TF\_AMBIT"); |
| Protected static String selenium\_url=System.getenv("MAT\_TF\_SELENIUM\_URL"); |
| Protected static String influxdb\_url=System.getenv("MAT\_TF\_INFLUXDB\_URL"); |
| Protected static String influxdb\_token=System.getenv("MAT\_TF\_INFLUXDB\_TOKEN"); |
| Protected static String influxdb\_bucket=System.getenv("MAT\_TF\_INFLUXDB\_BUCKET"); |
| Protected static String influxdb\_company=System.getenv("MAT\_TF\_INFLUXDB\_COMPANY"); |
| Protected static String selenium\_firefox\_driver=System.getenv("MAT\_TF\_SELENIUM\_FIREFOX\_DRIVER"); |
| Protected static String environment=System.getProperty("environment"); |
| Protected static String build\_id=System.getProperty("build\_id"); |
| Protected static String job\_name=System.getProperty("job\_name"); |
| Protected static String jira\_pk=System.getProperty("jira\_pk"); |
| Protected static String jira\_issue=System.getProperty("jira\_issue"); |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

Les variables de environment, build\_id, job\_name i les relacionades amb influx són necessàries per a la connexió del projecte amb Grafana.

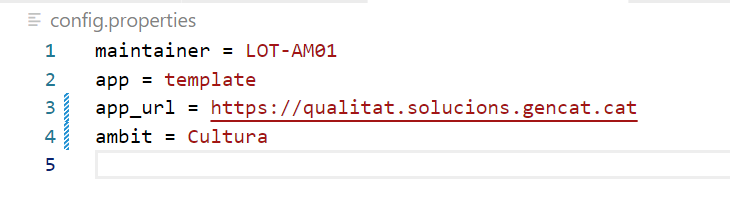
En l'apartat 5.2 es mostra com es realitza l'execució de la plantilla. En aquest exemple, no es realitzarà cap anomenada a les variables de Inlfux.

Diferents formes de aplicar les variables en els paràmetres d'entrada.

* Directament des d'un comandament en el terminal. (-D)

Example: *mvn clean test -Dselenium\_url="**http://localhost:4444/wd/hub"*

* A través del fitxer “config.properties” de la plantilla.



La lògica de la classe està formada de manera que si el tester inclou els paràmetres en l'arxiu de *config.properties*, no és necessari que els cridi en el moment de l'execució.

No obstant això, si en l'arxiu de *config.properties* està declarada una variable i de nou es diu en la terminal amb el comando -D, aquesta tindrà preferència.

És a dir, el flux del projecte serà mirar si algun paràmetre va ser manat amb el comando -D, en cas que no haver-hi res en -D, buscarà en el config.properties i en cas de no veure res en config.properties, reconecerá lavariable com nul·la.

Per exemple, en el fitxer de config.properties es defineix la variable d'app\_url.

En la terminal es pot manar simplement la instrucción:

*mvn clean test -Dselenium\_url="http://localhost:4444/wd/hub"*

En cas de manar per consola la variable app\_url, el projecte llegirà primer la variable definida amb el comando -D, ignorant la que està en *config.properties*.

*mvn clean test -Dselenium\_url="http://localhost:4444/wd/hub" -Dapp\_url="https://qualitat.solucions.gencat.cat"*

***Què és******InfluxDB?***

[**InfluxDB**](https://www.influxdata.com/partners/grafana/)és una base de dades de sèries temporals que generen mètriques de l'aplicació.  
**InfluxDB** s'integra amb **[Grafana](https://www.influxdata.com/partners/grafana/)**, que és una plataforma que permet el monitoratge i visualització de panells de control.

### **BaseTest.java**

La classe **BaseTest**.java pertany a la llibreria, i conté mètodes comuns (inciar el driver, tancar el driver...) vàlids per a qualsevol projecte que s'executaran abans de cada classe, abans de cada mètode, en iniciar la suite, etc.

|  |  |  |
| --- | --- | --- |
| **Anotació** | **Mètodes** | **Explicació** |
| @BeforeSuite | suiteInit() | Inicialitza el conjunt de proves. |
| @BeforeMethod | testInit(String browser, Method method) | Procés de configuració per a un cas de prova específic. |
| @AfterMethod | testQuit() | Atura el controlador web. |
| @AfterClass | testShutdown() | Neteja tots els recursos relacionats amb el cas de prova. |
| @AfterSuite | suiteShutdown() | Atura el conjunt de proves. |

### **Utils.java**

La classe Utils conté mètodes que poden ser necessaris en qualsevol projecte com accedir a l'aplicació, maximitzar la finestra, scroll, verificar un element...

|  |  |
| --- | --- |
| **Mètodes** | **Explicació** |
| gotoApp() | Accedir a la url de l'app a testar.  Aquest mètode diu el paràmetre app\_url. |
| maximize() | Maximitzar la finestra del navegador |
| getElement(By selector) | Obtenir l'element web d'un selector. |
| getElement(By selector, int timeout) | Obtenir l'element web d'un selector amb un temps d'espera. |
| scroll(int percent\_x, int percent\_y, int timeout) | Desplaça la finestra (en percentatge) amb un temps d'espera. |
| scrollToBottom(int timeout) | Desplaça la finestra cap a la part inferior (eix Y) amb un temps d'espera. |
| scrollToTop(int timeout) | Desplaceu la finestra cap a la part superior (eix Y) amb un temps d'espera. |
| step(String name) | Definiu un nou pas dins d'un cas de prova. |
| anotate(LogLevel level, String msg) | Afegeix una anotació dins del context del cas de prova actual.  Sent "level" l'estat del pas i "msg" el missatge explicatiu |
| screenshot(String caption) | Feu una captura de pantalla nova de l'àrea de visualització del navegador actual i adjunteu-la al context del cas de prova actual. |
| endTestAsOK(String browser, Method method) | Gestiona el final del cas de prova amb èxit. |
| endTestAsKO(String browser, Method method, Throwable e) | Gestiona el final del cas de prova amb error. |

### **BrowserOptions.java**

La classe BrowserOptions conté totes característiques pròpies per a cada navegador.

### **ExtentManager.java**

Aquesta classe defineix els mètodes els mètodes necessaris per a bolcar els resultats al document .html.

### **ResultSender.java i ExecutionListener.java**

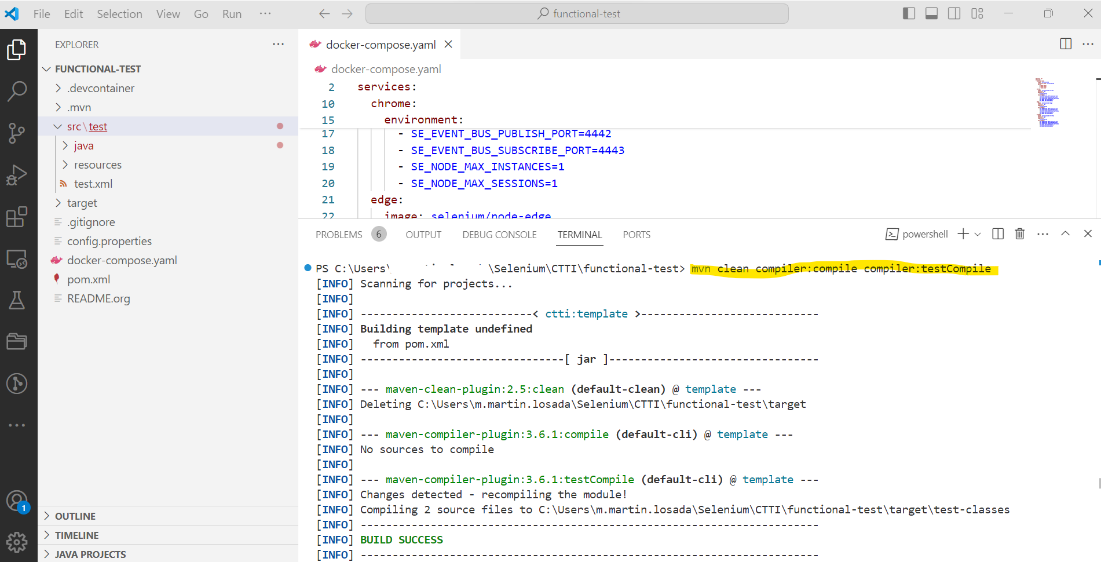
Aquestes classes són les que gestionen la connexió amb InfluxDB.

## **Compilar i executar la plantilla**

Com ja està Selenium Grid aixecat (veure pas 4), ja es pot executar la plantilla de prova, per a comprovar que tot funciona correctament.

Comprovar que el projecte compila correctament:

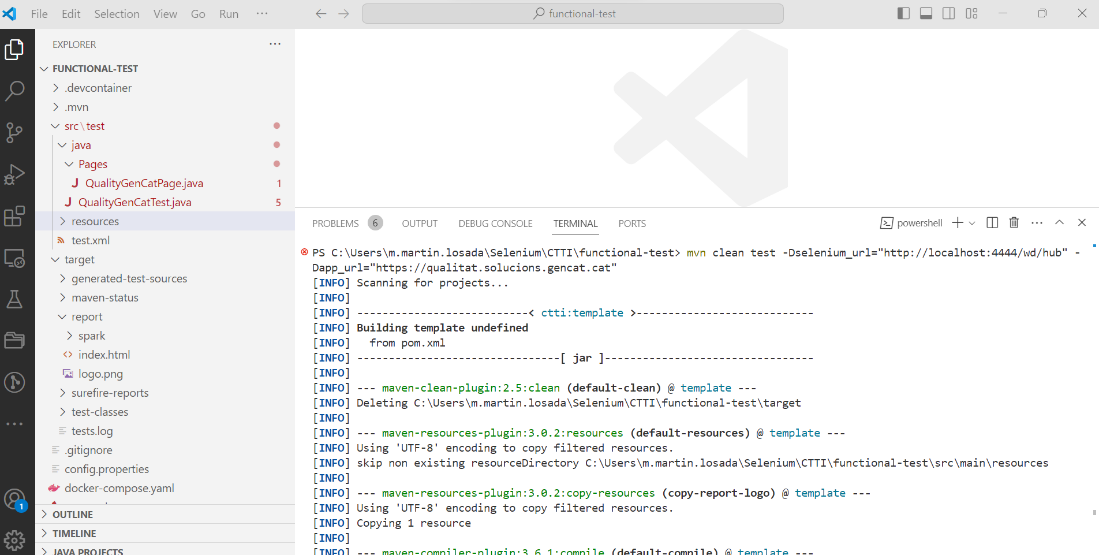
*mvn clean compiler:compile compiler:testCompile*

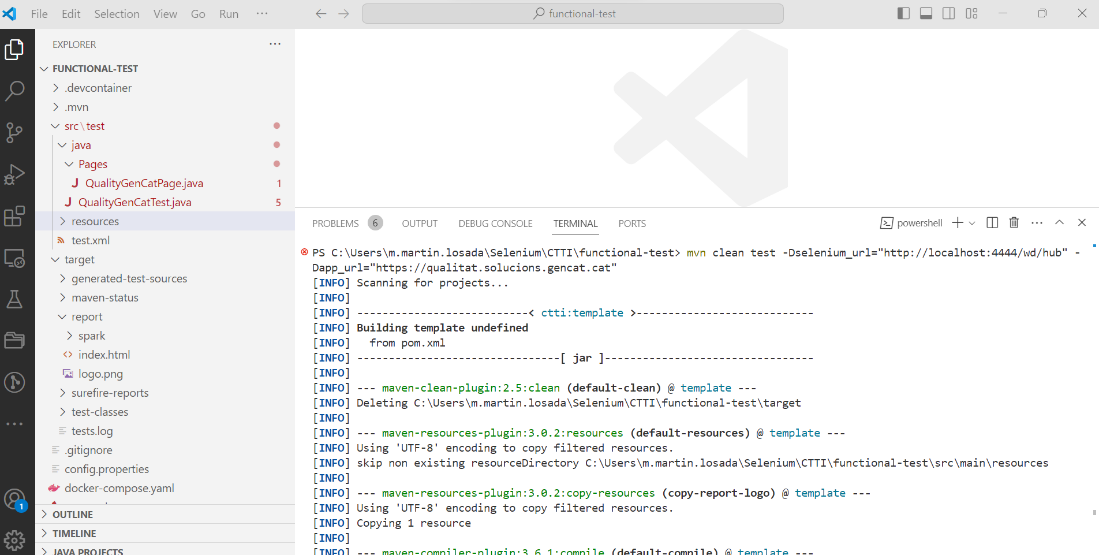


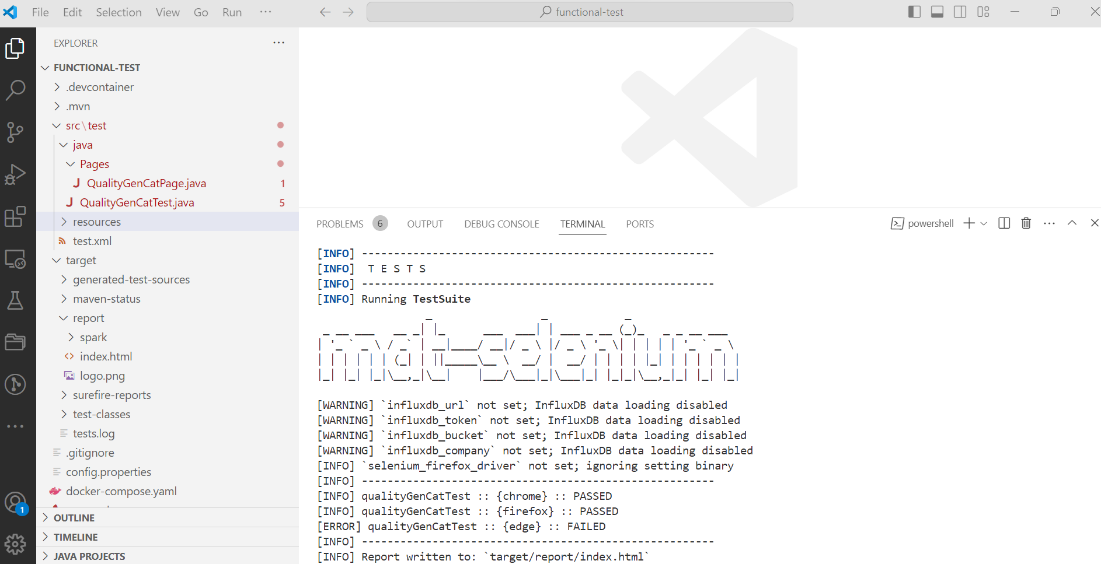
Executar el projecte:

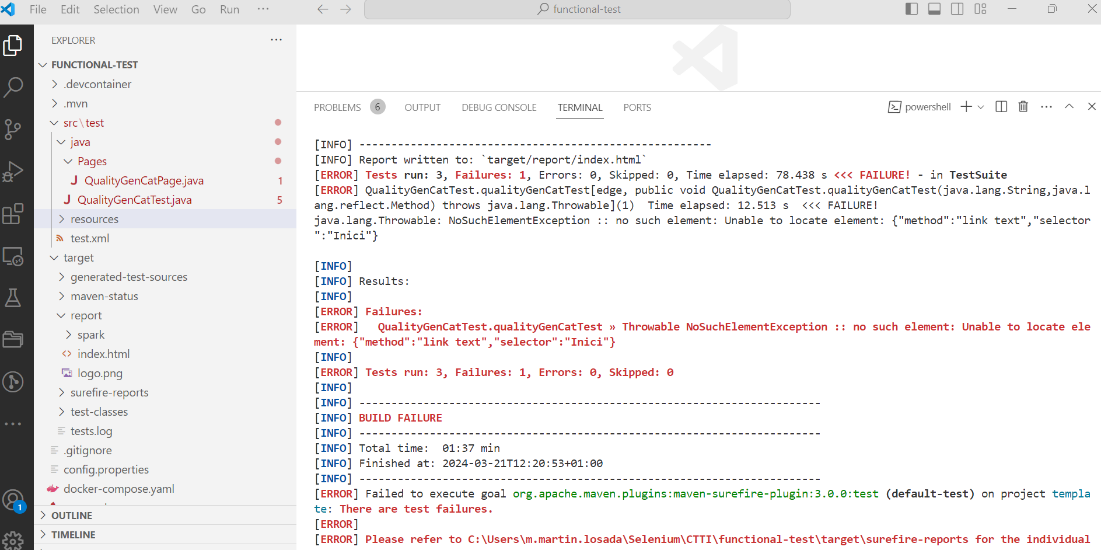
*mvn clean test -Dselenium\_url="http://localhost:4444/wd/hub" -Dapp\_url="https://qualitat.solucions.gencat.cat"*

*mvn clean test -Dselenium\_url="[url Selenium Grid]” -Dapp\_url="[Url de les proves]"*









# **Crear casos de proves**

El test d'exemple que es crearà és el següent:

1. Navega per [www.google.com](http://www.google.com)
2. Verificar imatge de “Google”.
3. En el cercador, cerca "Selenium".
4. Clic en l'opció de "Selenium" del desplegable.
5. Buscar la url de <https://www.selenium.dev/>
6. Clic en Selenium.

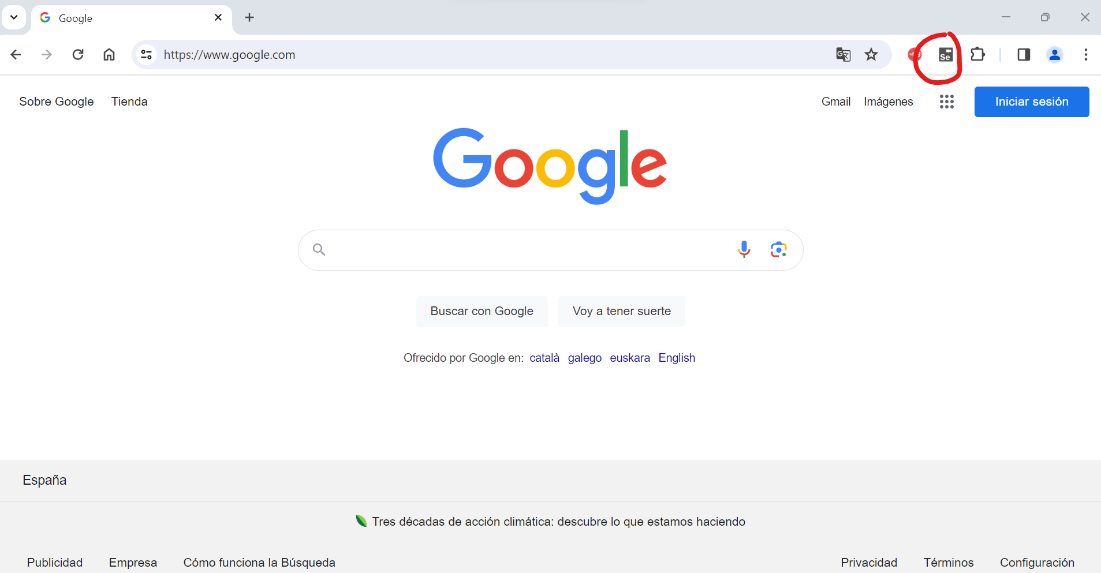
## **Definir Page Objet Models**

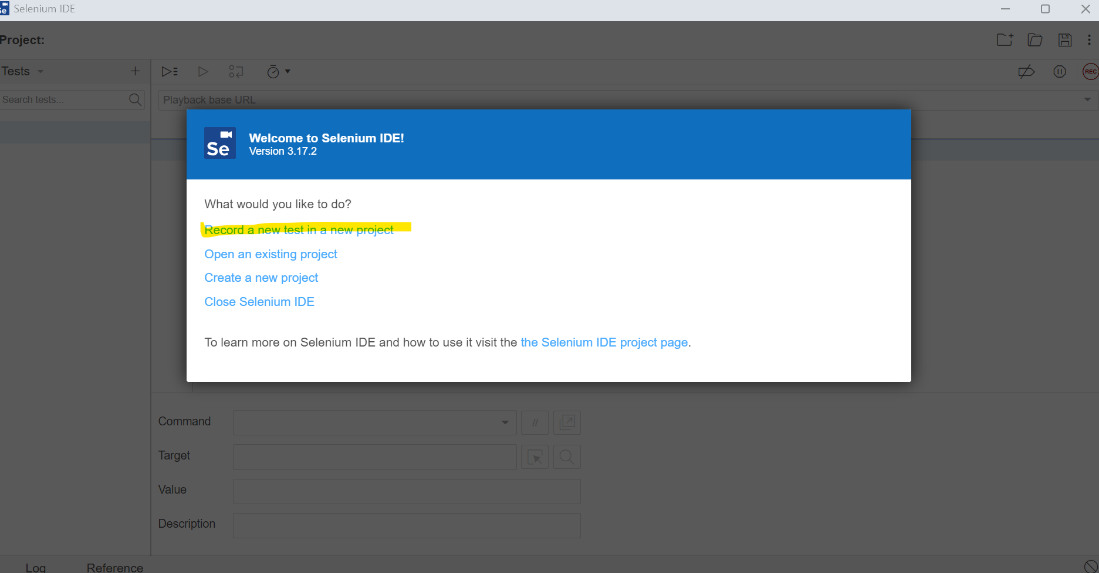
Per a crear els casos de prova, primer s'han de definir la "Page". Aquesta classe contindrà tots els elements específics necessaris per a les proves ([page object models](https://www.selenium.dev/documentation/test_practices/encouraged/page_object_models/)), per exemple, validar un text, clic en un element de la web, etc.

Realitzar una còpia de QualityGenCatPage.java i canviar el nom concorde a les teves proves- (GooglePage.java).

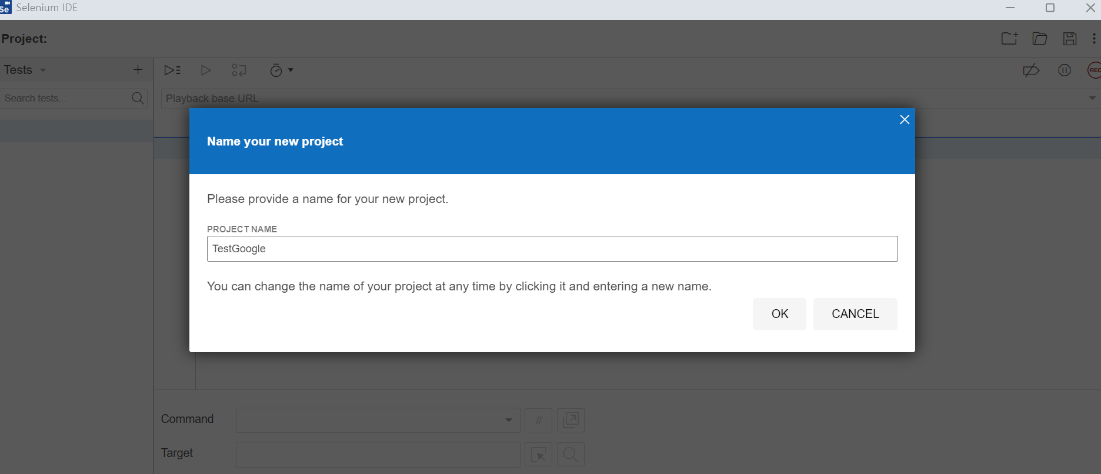


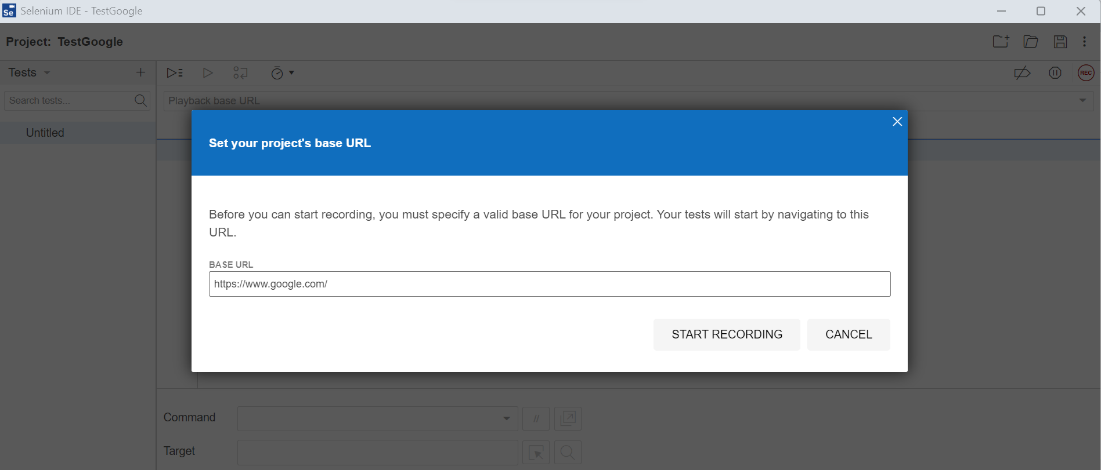
Per a definir els objectes localitzadors, es pot utilitzar l'extensió de Chrome de [Selenium IDE](https://www.selenium.dev/documentation/legacy/selenium_ide/). Aquesta extensió permet conèixer tots els possibles localitzadors de cada element de la pàgina.



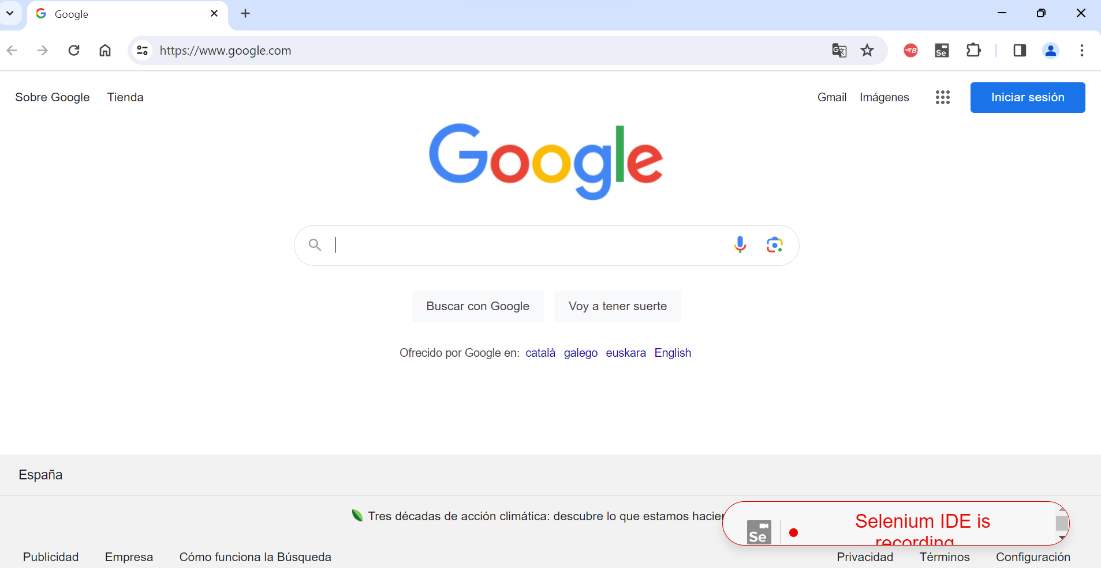


No és necessari guardar aquest projecte de Selenium IDE. Aquesta extensió és útil per a conèixer quins tipus d'elements hem d'incloure en nuetra classe GooglePage.java.

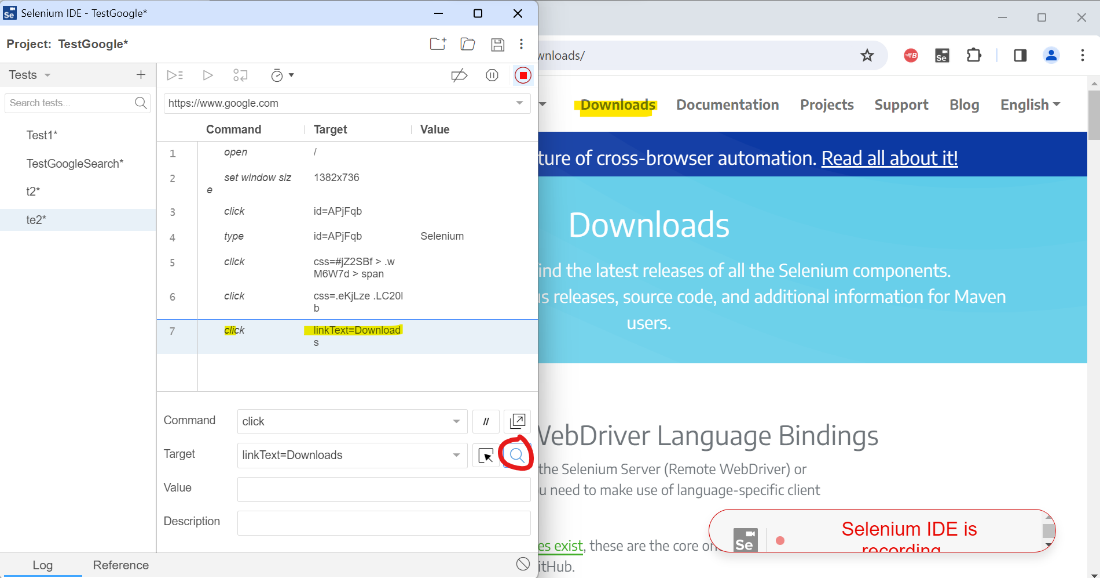




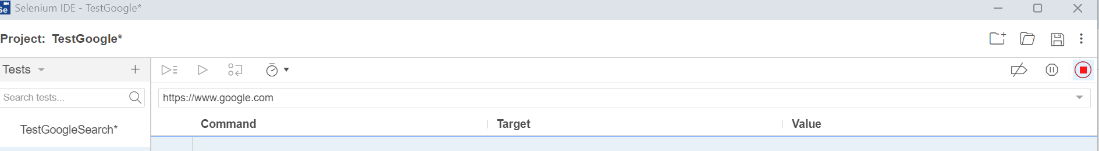
S'aixeca un navegador que registrarà les accions que es realitzin.



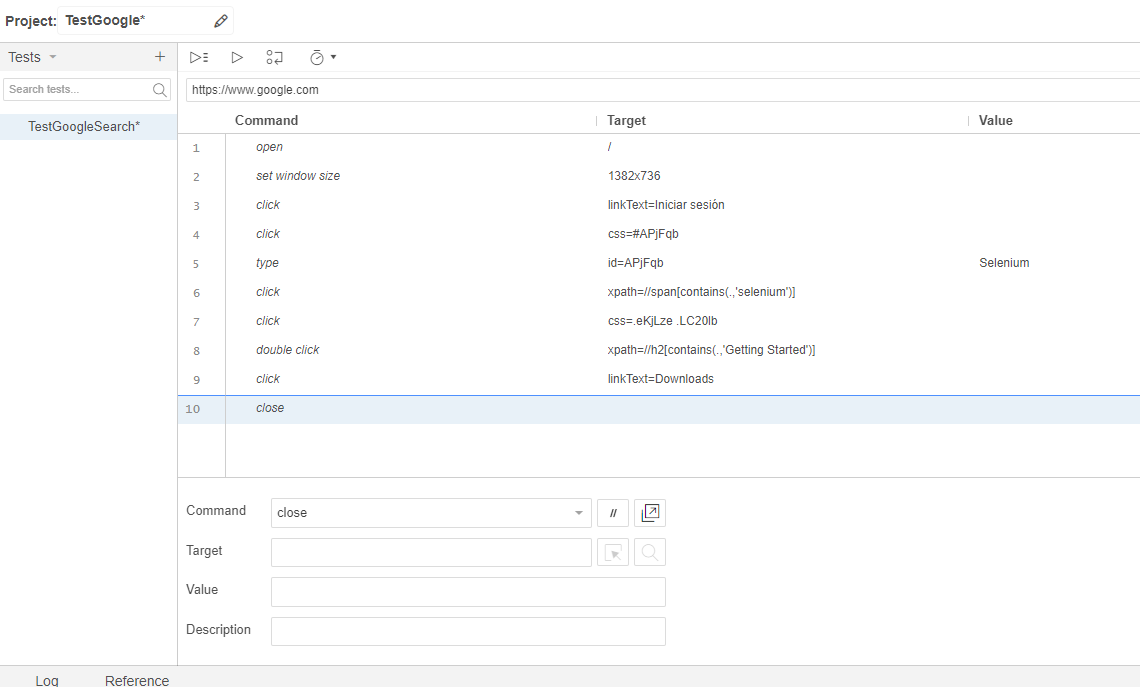
Mentre la gravació es trobi activa, es pot comprovar amb la lupa com és element identificat per Selenium IDE en el propi navegador.



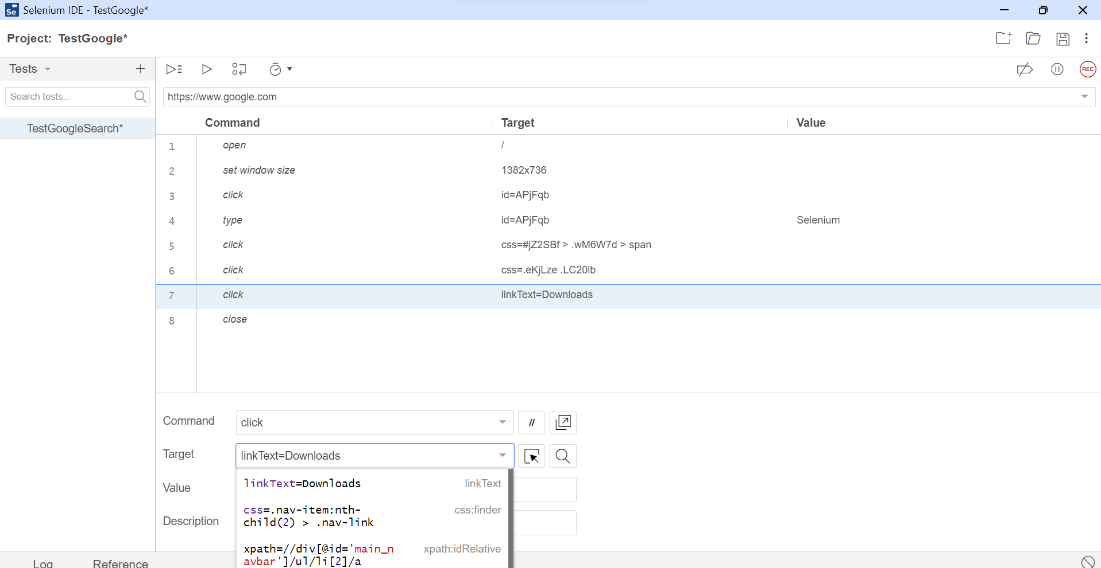
Una vegada executades totes les accions necessàries per al test, es per a la gravació.



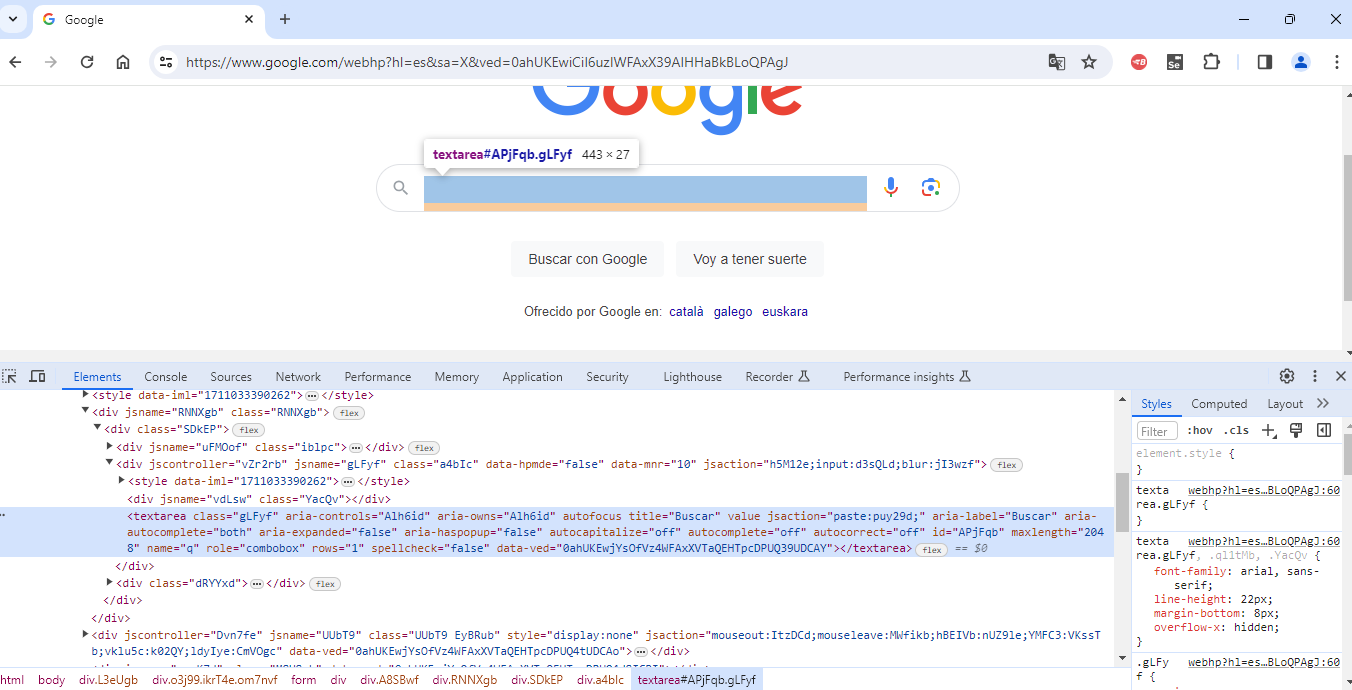
I Selenium IDE mostra totes les accions realitzades:



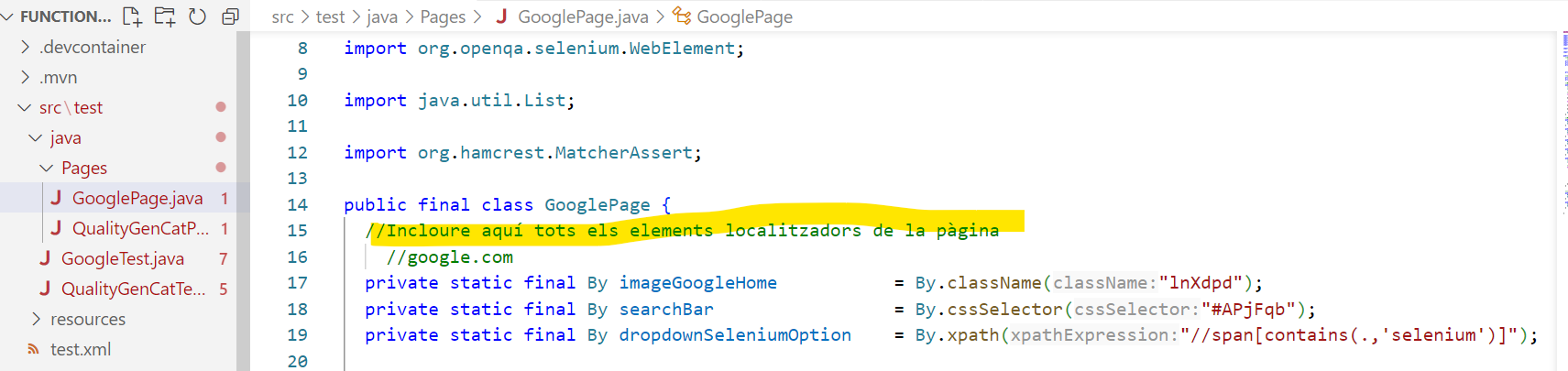
Cadascuna de les accions realitzades es grava amb una mena d'objecte. En el desplegable, és possible veure altres maneres d'identificació de l'element.



Tots aquests elements també es poden identificar amb el Inspect del propi navegador.



Coneixent els elements, ja es pot definir en GooglePage.java les variables estàtiques:



public final class GooglePage extends BaseTest {

//Incloure aquí tots els elements localitzadors de la pàgina

    //google.com

private static final By imageGoogleHome           = By.className("lnXdpd");

  private static final By searchBar                 = By.cssSelector("#APjFqb");

Ara cal crear els mètodes necessaris per a l'execució del test, seguint la mateixa estructura que la plantilla.



//Mètodes relacionats amb la pàgina de Google

  public static void checkImageGoogleHome( ) {

    BaseTest.getDriver().findElements(imageGoogleHome);

    Utils.anotate(Utils.LogLevel.PASS, "Google Image Home");

  }

  public static String clickAndSearchBar (String somethingToSearch) {

    Utils.getElement(searchBar).sendKeys(somethingToSearch);

    Utils.anotate(Utils.LogLevel.PASS, "Search Bar is clickable and available to send keys");

    return somethingToSearch;

  }

  public static void clickDropdownOption(String label) {

    //Elements del resultat de la cerca (dropdown)

    List<WebElement> elements = BaseTest.getDriver().findElements(By.cssSelector("div.pcTkSc[aria-label='" + label + "']"));

    //Iteració - clic

    for (WebElement element : elements) {

        element.click();

        break;

    }

  }

  public static void clickElementByHref(String href) {

    //Elements de la cerca trobats en Google

    List<WebElement> elements = BaseTest.getDriver().findElements(By.cssSelector("span[jscontroller='msmzHf']"));

    //Iteració

    for (WebElement element : elements) {

      WebElement anchor = element.findElement(By.tagName("a")); //Buscar <a>

      String elementHref = anchor.getAttribute("href"); //Obtenir valor de href en <a>

        //Clic quan es encuetre href

        if (elementHref.equals(href)) {

          anchor.click();

          break;

        }

    }

  }

|  |  |  |
| --- | --- | --- |
| **Tipus de mètodes** | **Mètodes** | **Explicació** |
| Comprovar element | checkImageGoogleHome() | Cerca d'un element de la web. |
| Clic i escriure un text | clickAndSearchBar() | En la barra de cerca, escriure un text. |
| Clic | clickDropdownOption() | Després de buscar "Selenium", Clickar en l'opció de Selenium del resultat mostrat |
| Buscar un element concret i clickar | ClickElementByHref() | Primer es fa una cerca de tots els elements HTML trobats en la pàgina. Després es busca el element que contingui el href que es necessita per al test i finalment es fa clic en ell. |

En els mètodes checkImageGoogleHome() i clickDropdownOption() és necessari cridar al mètode getDriver() de la classe BaseTests.  
Per a això, és necessari que la classe GooglePage estengui de BaseTest.

## **Definir Test**

De la mateixa manera que amb la creació dels Page Objet Models, es crea una còpia del test QualityGenCatTest.java concorde a les nostres proves (GoogleTest.java).

Es recomana que tots els test deuen tinguin la terminació “Test” per a la seva millor identificació.

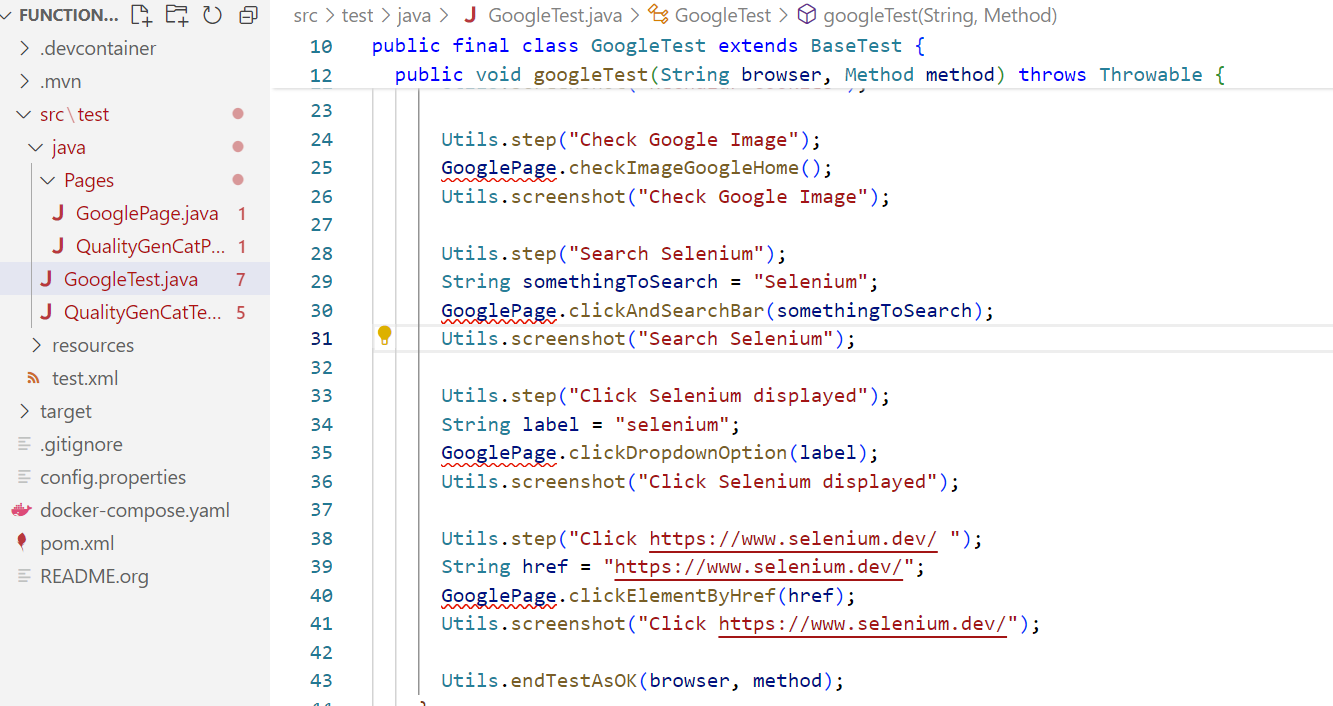
El test estén de la classe BaseTest. Aquesta classe forma part de la llibreria, i és on es defineixen els mètodes comuns per a totes les proves: iniciar el driver, llegir els paràmetres, generar els resultats HTML...



Els primers passos del test seran comuns a totes les proves, i no és necessari modificar-los. Aquests em´tots estàs definidios en la classe Utils (veure apartat 5.1.3). Aquesta classe Utils també forma part de la plantilla i recull mètodes comuns que poden ser d'utilitat en qualsevol projecte (obrir navegador, maximitzar, scroll...).

Aquests mètodes són:

|  |  |
| --- | --- |
| **Mètode** | **Explicació** |
| Utils.step("Enter website"); | Generar un pas en el report HTML. |
| Utils.gotoApp(); | Obrir el navegador i accedeix a la url donada. |
| Utils.maximize(); | Maximitzar el navegador. |
| Utils.anotate(Utils.LogLevel.PASS, "Website's homepage is accessible"); | Mostrar el resultat en el report HTML. |
| Utils.screenshot("Homepage"); | Realitzar una captura de pantalla. |



Després s'afegeixen la resta dels passos necessaris per a la prova. En aquest cas, es creessin anomenades als mètodes (definits en GooglePage.java) que compleixin les següents accions:

1. Navega per [www.google.com](http://www.google.com)
2. Verificar imatge de “Google”.
3. En el cercador, cerca "Selenium".
4. Clic en l'opció de "Selenium" del desplegable.
5. Buscar la url de <https://www.selenium.dev/>
6. Clic en Selenium.

Aquest passos estan dissenyats de manera que el test generarà una fallada a l'inici. En el apratado 8.1 s'analitzarà el motiu d'aquest.

D'aquesta manera, el test quedaria així:

public final class GoogleTest extends BaseTest {

  @Test @Parameters(value={"browser"})

  public void googleTest(String browser, Method method) throws Throwable {

    try {

      Utils.step("Enter website");

      Utils.gotoApp();

      Utils.maximize();

      Utils.anotate(Utils.LogLevel.PASS, "Website's homepage is accessible");

      Utils.screenshot("Homepage");

Utils.step("Check Google Image");

      GooglePage.checkImageGoogleHome();

      Utils.screenshot("Check Google Image");

      Utils.step("Search Selenium");

      String somethingToSearch = "Selenium";

      GooglePage.clickAndSearchBar(somethingToSearch);

      Utils.screenshot("Search Selenium");

      Utils.step("Click Selenium displayed");

      String label = "selenium";

      GooglePage.clickDropdownOption(label);

      Utils.screenshot("Click Selenium displayed");

      Utils.step("Click https://www.selenium.dev/ ");

      String href = "https://www.selenium.dev/";

      GooglePage.clickElementByHref(href);

      Utils.screenshot("Click https://www.selenium.dev/");

      Utils.endTestAsOK(browser, method);

    }

    catch (Exception | AssertionError e) { Utils.endTestAsKO(browser, method, e); }

  }

}

Cal tenir en compte que els mètodes de "clickAndSearchBar", “clickDropdownOption” i "clickElementByHref" estan definits per String.

És per això, que per a poder executar aquests dos mètodes cal declarar prèviament el String.

## **Definir test.xml**

Aquest arxiu .xml és el que s'encarrega de recopilar tots els test del projecte amb els seus caracterísiticas. És a dir, aquí es definirà en quin navegador s'executarà [cada test](https://maven.apache.org/surefire/maven-surefire-plugin/examples/testng.html#running-testnames-in-test-tag) i l'anomenada a la classe que defineix [el reporti HTML](https://maven.apache.org/surefire/maven-surefire-plugin/examples/testng.html#using-custom-listeners-and-reporters).

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite thread-count="3" name="template" parallel="tests">

  <listeners>

    <listener class-name="cat.gencat.mat.ExecutionListener" />

  </listeners>

<test name="GoogleTest.Chrome" parallel="methods">

    <parameter name="browser" value="chrome" />

    <classes>

      <class name="GoogleTest"></class>

    </classes>

  </test>

<test name="GoogleTest.Edge" parallel="methods">

    <parameter name="browser" value="edge" />

    <classes>

      <class name="GoogleTest"></class>

    </classes>

  </test>

  <test name="GoogleTest.Firefox" parallel="methods">

    <parameter name="browser" value="firefox" />

    <classes>

      <class name="GoogleTest"></class>

    </classes>

  </test>

</suite>

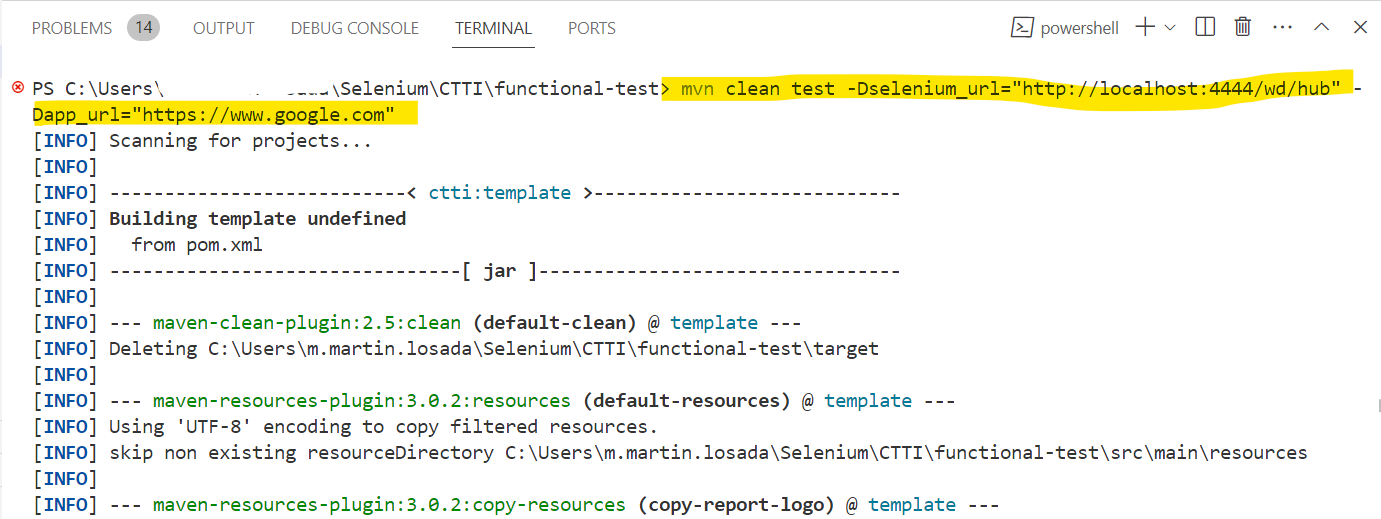
# **Executar les proves des de Visual Studio Code**

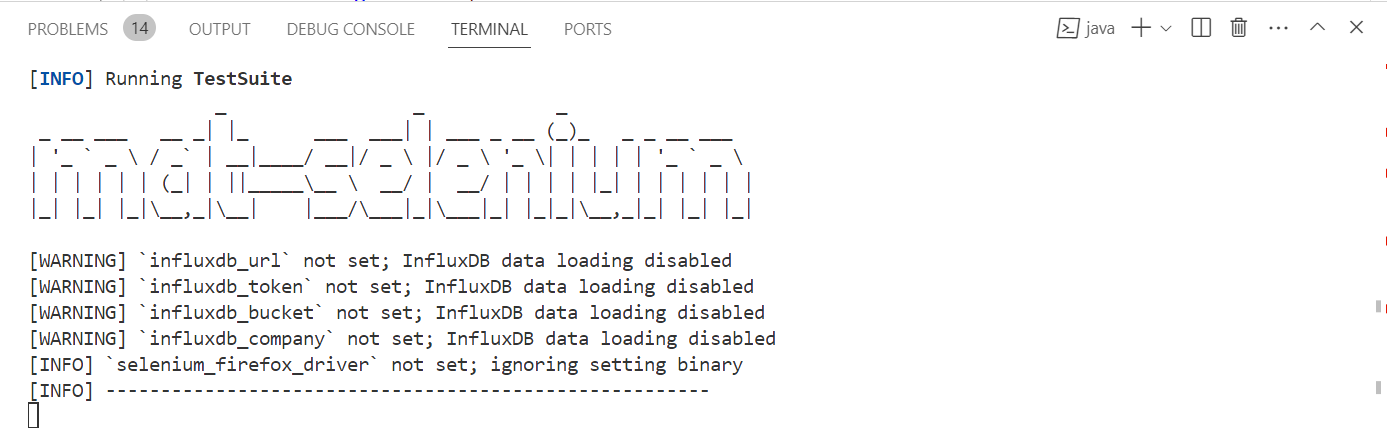
Una vegada definits els test i implementats en el test.xml, ja es pot executar les proves des de la pròpia consola del Visual Studio Code, a través de la instrucció definida en l'apartat 5:

*mvn clean test -Dselenium\_url="[url Selenium Grid]” -Dapp\_url="[Url de les proves]"*

*mvn clean test -Dselenium\_url="http://localhost:4444/wd/hub" -Dapp\_url="https://www.google.com"*

Comença l'execució del test:



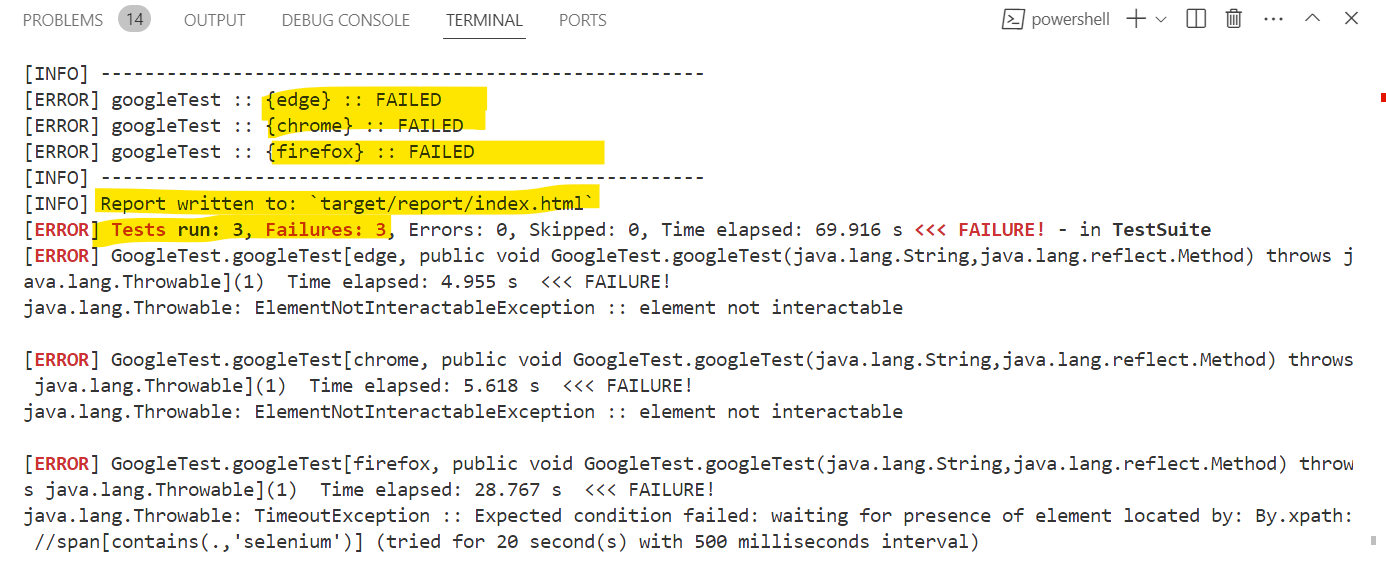


Per consola, s'anirà mostrant tota la informació definida en la llibreria i en els tests.

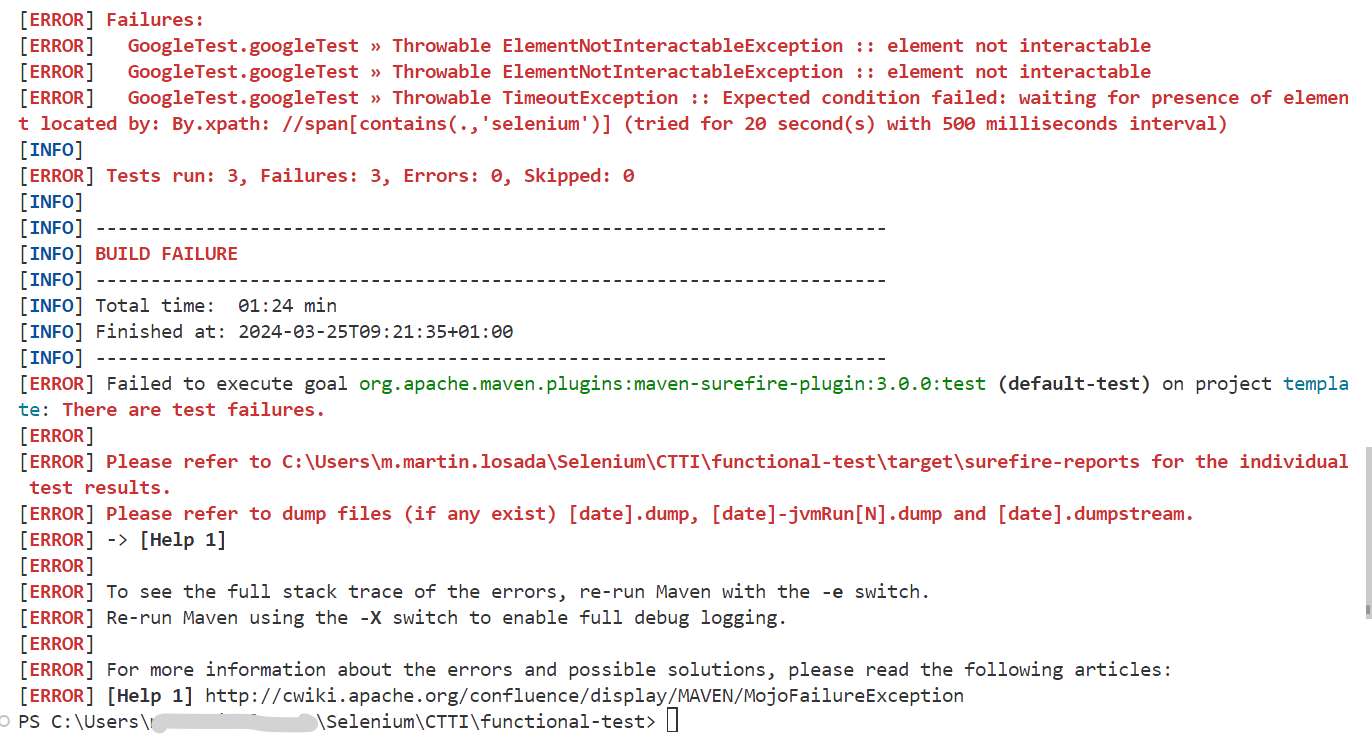
En aquest cas, els WARNING relatius a InfluxDB indiquen que cap paràmetre de InfluxDB va ser definit, ja que aquests paràmetres solament s'activaran quan es llancin les proves en un entorn no local.

Quan finalitza cada test, mostra l'estat de cadascun d'ells i el navegador on va ser executat.

També s'indica que el reporti està disponible en la localització de target/report/index.html

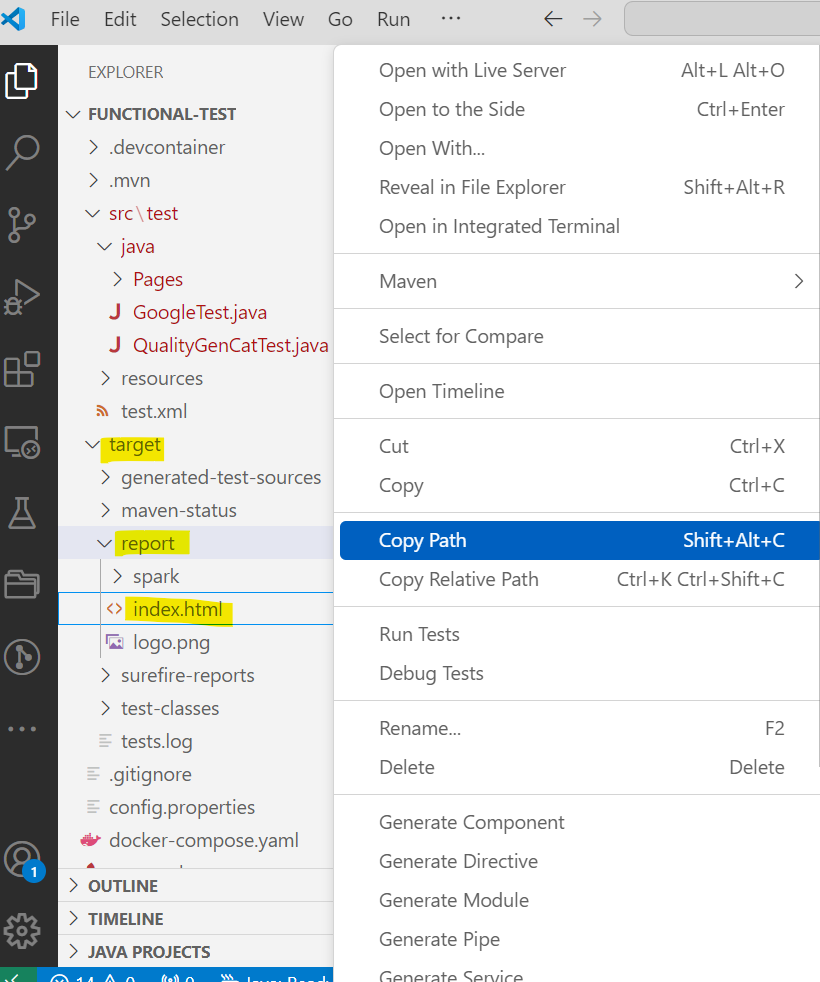


Finalment, també mostra una breu descripció del motiu de la fallada.



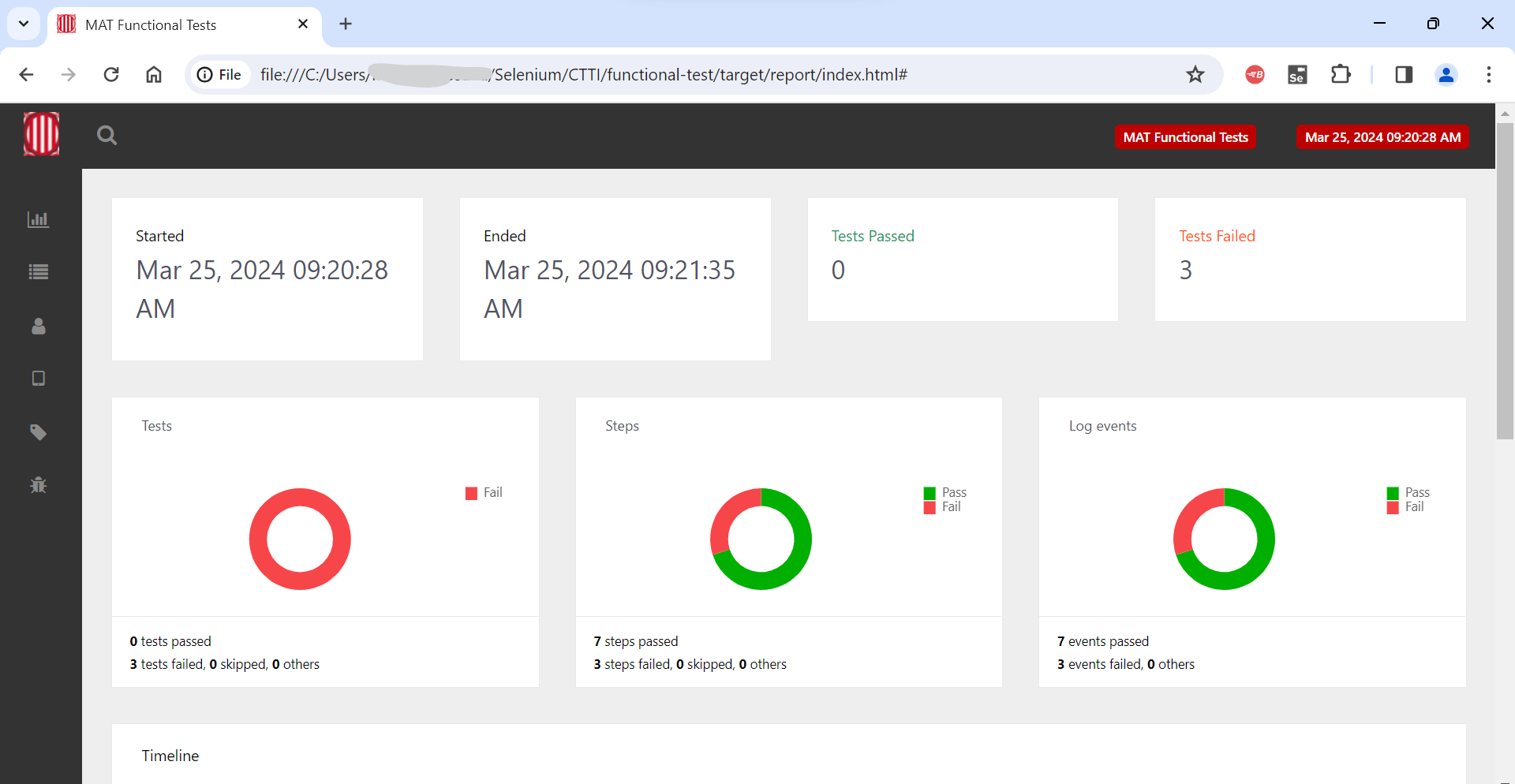
# **Anàlisi de resultats en el report HTML.**

Una vegada acabada l'execució, es genera automàticament un reporti amb els resultats en format *.html*. Aquest es troba guardat en *target/report/index.html.*



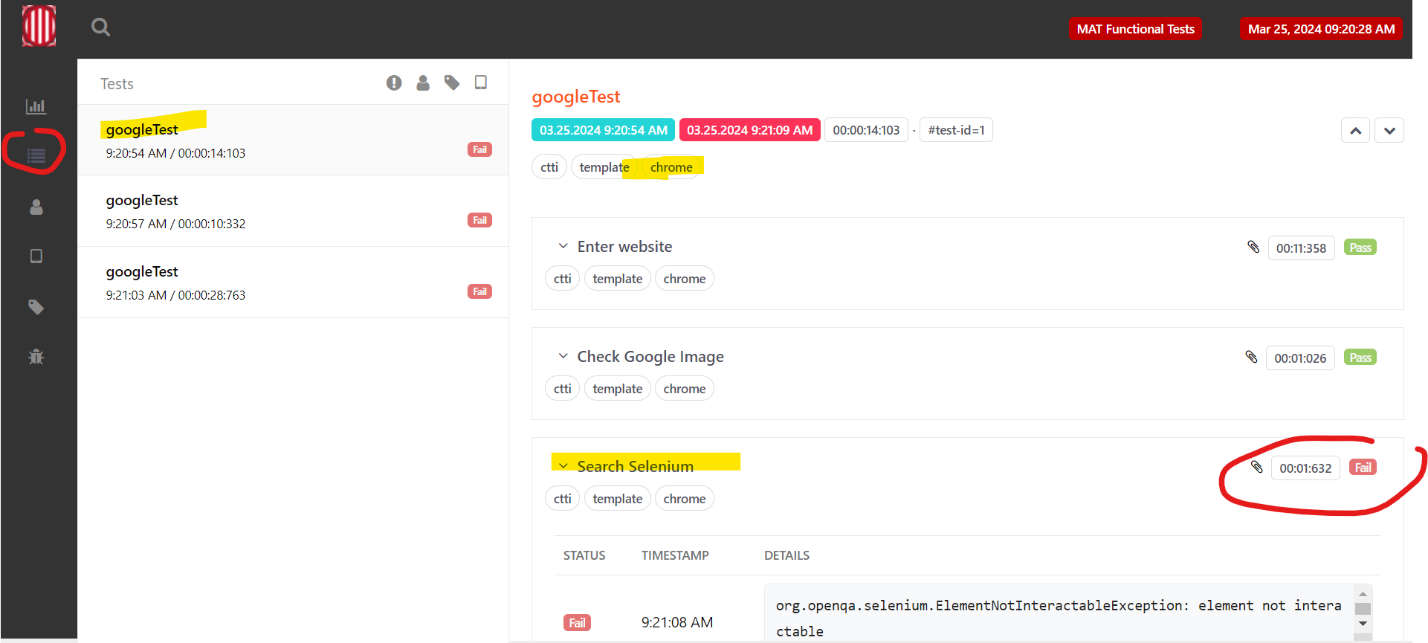
Es copia la direcció en un navegador, per a poder visualitzar els resultats de forma més còmoda.

En una primera visió, es pot veure el resum de l'execució i els casos passats o fallats.



Accedint a la llista dels test, es pot veure cada test executat i dins de cadascun d'ells, els passos realitzats (indicant si van ser passats o fallats i les seves captures de pantalla).

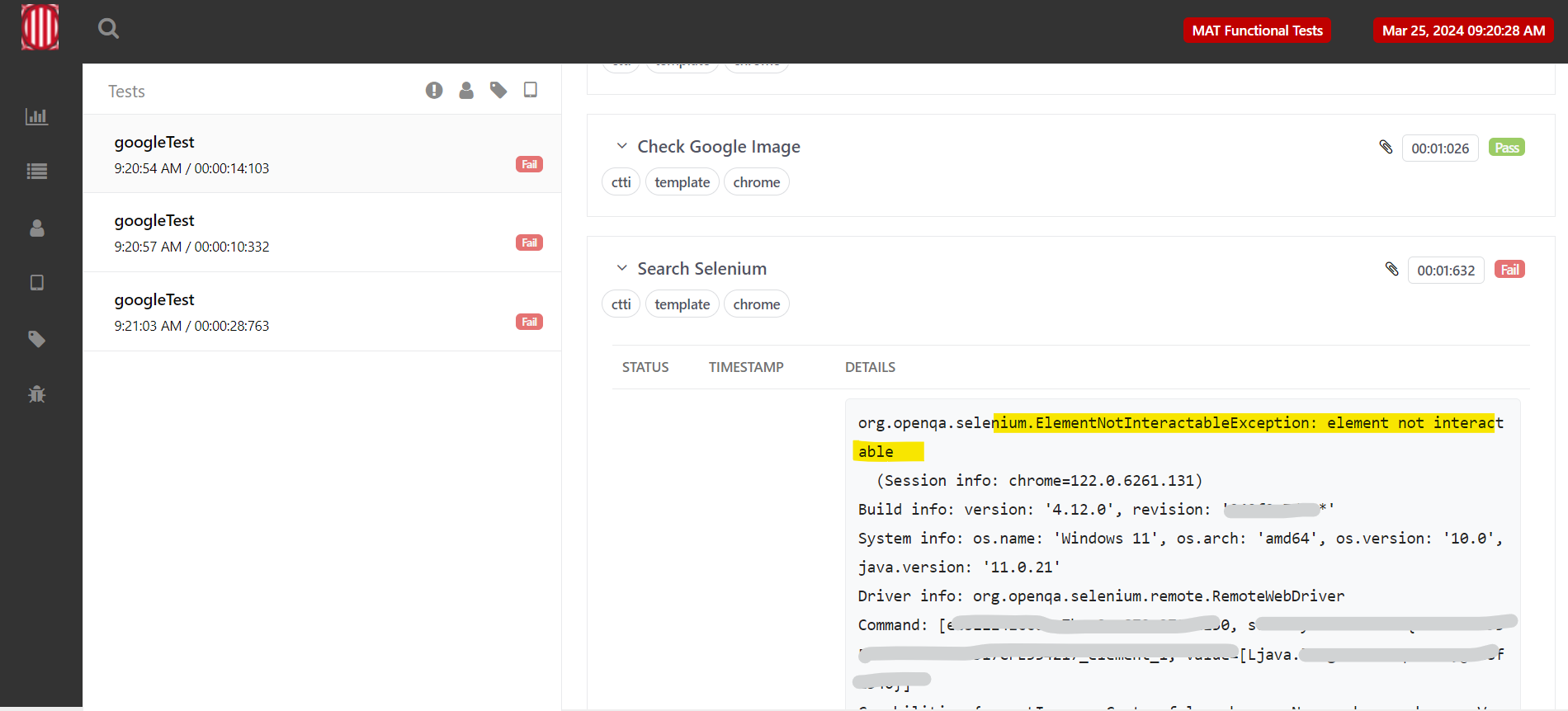
Dins del detall del pas, es pot veure el motiu de l'error.

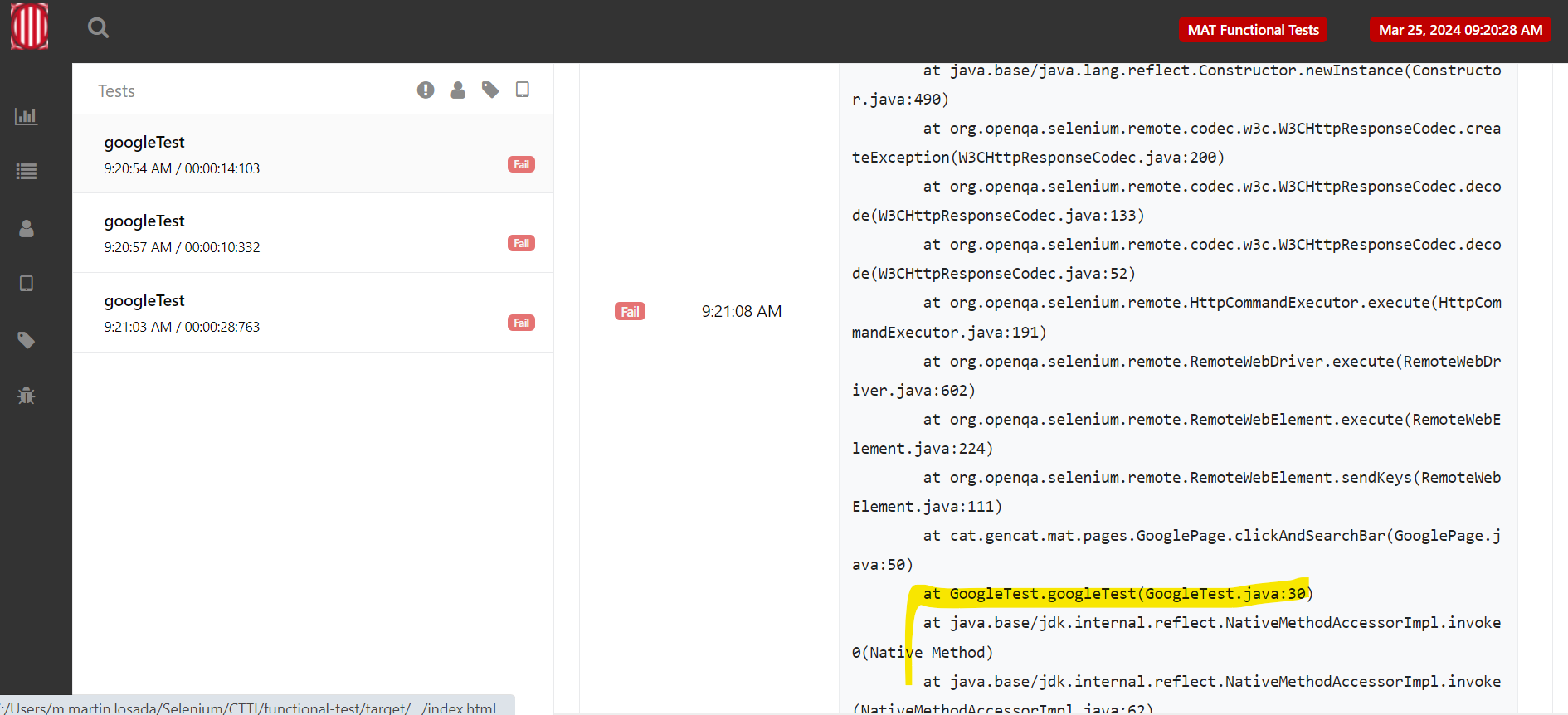


## **Anàlisi de l'error**

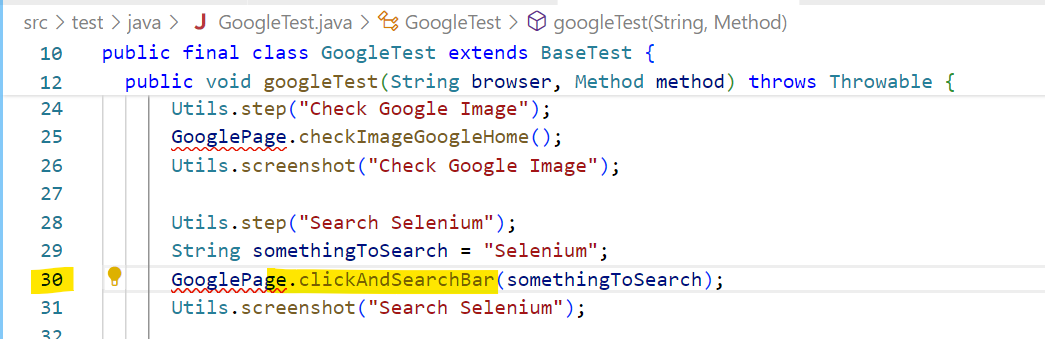
L'error ocorre en el pas de "Search Selenium".

Desplegant el detall de l'error, es pot veure que no es pot interactuar amb l'element, i que es troba en la línia 30 del test GoogleTest.java.





És a dir, l'error ocorre en l'anomenada al mètode clickAndSearchBar().

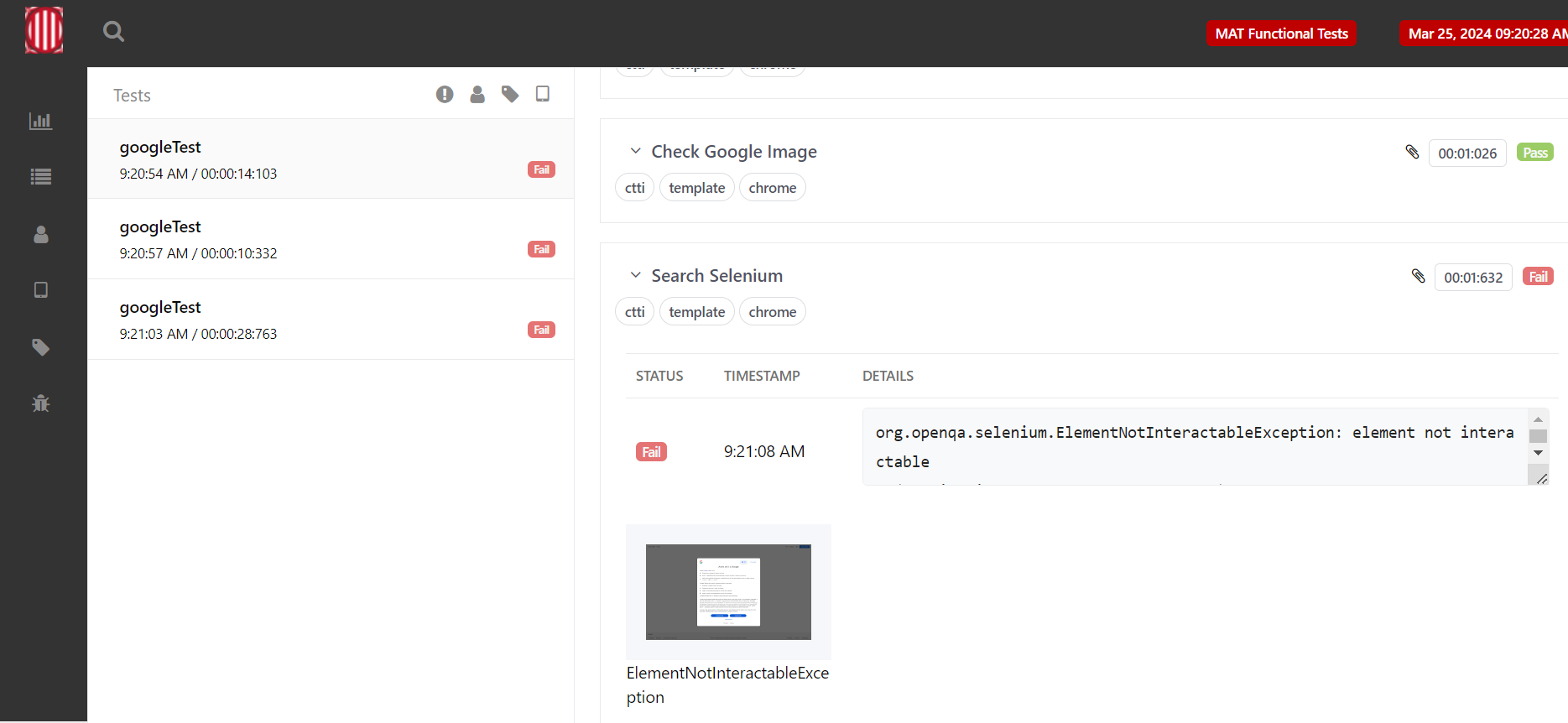


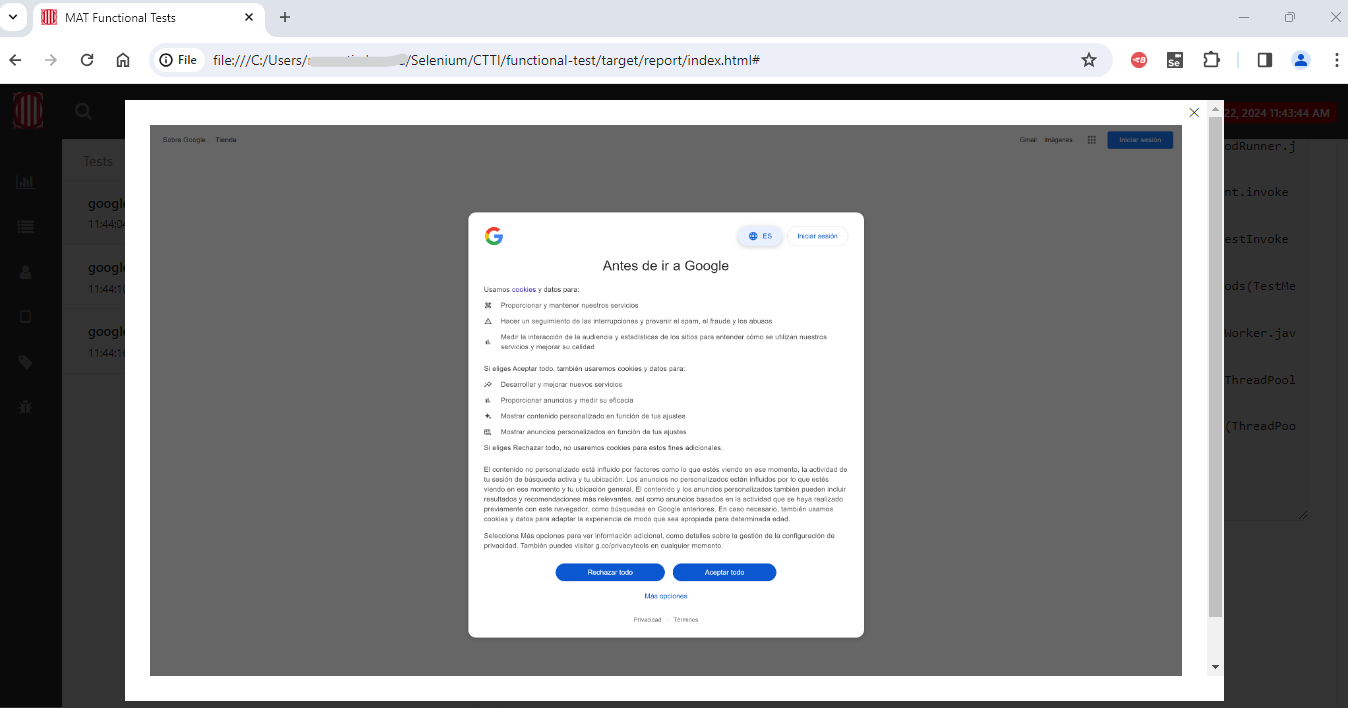
Tal com s'explica en l'apartat 6.1, aquest mètode busca un element i el compara amb un text.

|  |  |  |
| --- | --- | --- |
| **Tipus de mètodes** | **Mètodes** | **Explicació** |
| Clic i escriure un text | clickAndSearchBar() | En la barra de cerca, escriure un text. |

Si no es troba l'element o no es pot interactuar amb ell per algun motiu, el test falla.

En el index.html també es pot veure la captura de pantalla del pas, per a entendre millor el motiu de la fallada.



  
  
En la captura de pantalla es veu que la Home no va ser oberta correctament, ja que s'obre un pop up per a Acceptar o Rebutjar les cookies (això ocorre perquè les execucions es realitzen en un navegador en ocult).

És per això, que en executar el mètode clickAndSearchBar() no pot veure l'element correctament.

### **Solucionar l'error**

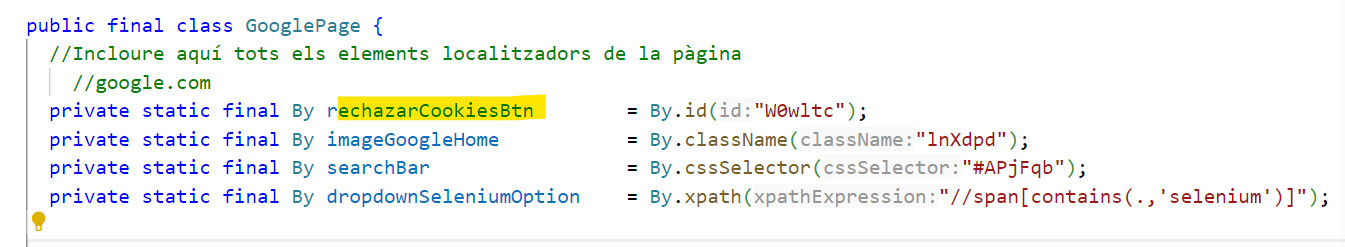
Per a solucionar aquest error, s'hauria de crear un mètode que Accepti o Rebutgi les cookies en cas que es desplegui el pop up.

#### **Modificar el POM**

Seguint els passos realitzats en l'apartat 6 de creació de casos de proves, primer s'ha de definir el POM relacionat amb el pop up.

Utilitzant qualsevol navegador en ocult, es pot simular el pop up de les Cookies mostrat en l'execució. Amb l'ajuda del Inspect, es pot obtenir l’id dels botons d'Acceptar o Rebutjar.





I s'afegeix un mètode que primer comprovi si existeix el pop up de les cookies, i en cas que existeixi, faci clic a Rebutjar.



Per tant, ara hi ha aquests mètodes:

|  |  |  |
| --- | --- | --- |
| **Tipus de mètodes** | **Mètodes** | **Explicació** |
| Comprovar element i si existeix, realitzar una acció. | checkAndClickRechazarCookiesBtn() | Comprovar pop up i si existeix fer clic. |
| Comprovar element | checkImageGoogleHome() | Cerca d'un element de la web. |
| Comprovar text | getSeleniumOption() | Cerca d'un element de la web i comprovar el seu text. |
| Clic i escriure un text | clickAndSearchBar() | En la barra de cerca, escriure un text. |
| Clic | ClickDropdownSeleniumOption() | Després de buscar "Selenium", Clickar en l'opció de Selenium del resultat mostrat |
| Buscar un element concret i clickar | ClickElementByHref() | Primer es fa una cerca de tots els elements HTML trobats en la pàgina. Després es busca el element que contingui el href que es necessita per al test i finalment es fa clic en ell. |

D'aquesta manera, la classe GooglePage.java queda d'aquesta manera:

public final class GooglePage {

  //Incloure aquí tots els elements localitzadors de la pàgina

    //google.com

private static final By rechazarCookiesBtn        = By.id("W0wltc");

  private static final By imageGoogleHome           = By.className("lnXdpd");

  private static final By searchBar                 = By.cssSelector("#APjFqb");

private static final By dropdownSeleniumOption    = By.xpath("//span[contains(.,'selenium')]");

//Mètodes relacionats amb la pàgina de Google

public static void checkAndClickRechazarCookiesBtn( ) {

    //Guardar tots els elements que contingui l'id W0wltc (Reject btn)

    List<WebElement> element = BaseTest.getDriver().findElements(By.id("W0wltc"));

    //Si algun element existeix, fer clic en el primer (get(0)). En aquest cas només apareixeria un element.

    if (!element.isEmpty()) {

      element.get(0).click();

    }

    try {

      Thread.sleep(5000); // Pausa de 2 segundos (ajusta el tiempo según sea necesario)

    } catch (InterruptedException e) {

      e.printStackTrace();

    }

    Utils.anotate(Utils.LogLevel.PASS, "Cookies Rechazadas");

  }

  public static void checkImageGoogleHome( ) {

    BaseTest.getDriver().findElements(imageGoogleHome);

    Utils.anotate(Utils.LogLevel.PASS, "Google Image Home");

  }

  public static String clickAndSearchBar (String somethingToSearch) {

    Utils.getElement(searchBar).sendKeys(somethingToSearch);

    Utils.anotate(Utils.LogLevel.PASS, "Search Bar is clickable and available to send keys");

    return somethingToSearch;

  }

public static void getSeleniumOption() {

    MatcherAssert.assertThat(Utils.getElement(dropdownSeleniumOption, 20).getText(), Matchers.equalTo("Selenium"));

    Utils.anotate(Utils.LogLevel.PASS, "Text is correct");

  }

  public static void clickDropdownSeleniumOption() {

    Utils.getElement(dropdownSeleniumOption).click();

    Utils.anotate(Utils.LogLevel.PASS, "Selenium option is clickable");

  }

  public static void clickElementByHref(String href) {

    //Elements de la cerca trobats en Google

    List<WebElement> elements = BaseTest.getDriver().findElements(By.cssSelector("span[jscontroller='msmzHf']"));

    //Iteració

    for (WebElement element : elements) {

      WebElement anchor = element.findElement(By.tagName("a")); //Buscar <a>

      String elementHref = anchor.getAttribute("href"); //Obtenir valor de href en <a>

        //Clic quan es encuetre href

        if (elementHref.equals(href)) {

          anchor.click();

          break;

        }

    }

  }

#### **Modificar el Test**

Després d'iniciar el navegador, i abans de cridar al mètode de checkImageGoogleHome(), cal cridar al mètode de checkAndClickRechazarCookiesBtn().

public final class GoogleTest extends BaseTest {

  @Test @Parameters(value={"browser"})

  public void googleTest(String browser, Method method) throws Throwable {

    try {

      Utils.step("Enter website");

      Utils.gotoApp();

      Utils.maximize();

      Utils.anotate(Utils.LogLevel.PASS, "Website's homepage is accessible");

      Utils.screenshot("Homepage");

Utils.step("Rechazar Cookies");

      GooglePage.checkAndClickRechazarCookiesBtn();

      Utils.screenshot("Rechazar Cookies");

      Utils.step("Check Google Image");

      GooglePage.checkImageGoogleHome();

      Utils.screenshot("Check Google Image");

      Utils.step("Search Selenium");

      String somethingToSearch = "Selenium";

      GooglePage.clickAndSearchBar(somethingToSearch);

      Utils.screenshot("Search Selenium");

Utils.step("Click Selenium displayed");

      String label = "selenium";

      GooglePage.clickDropdownOption(label);

      Utils.screenshot("Click Selenium displayed");

      Utils.step("Click https://www.selenium.dev/ ");

      String href = "https://www.selenium.dev/";

      GooglePage.clickElementByHref(href);

      Utils.screenshot("Click https://www.selenium.dev/");

      Utils.endTestAsOK(browser, method);

    }

    catch (Exception | AssertionError e) { Utils.endTestAsKO(browser, method, e); }

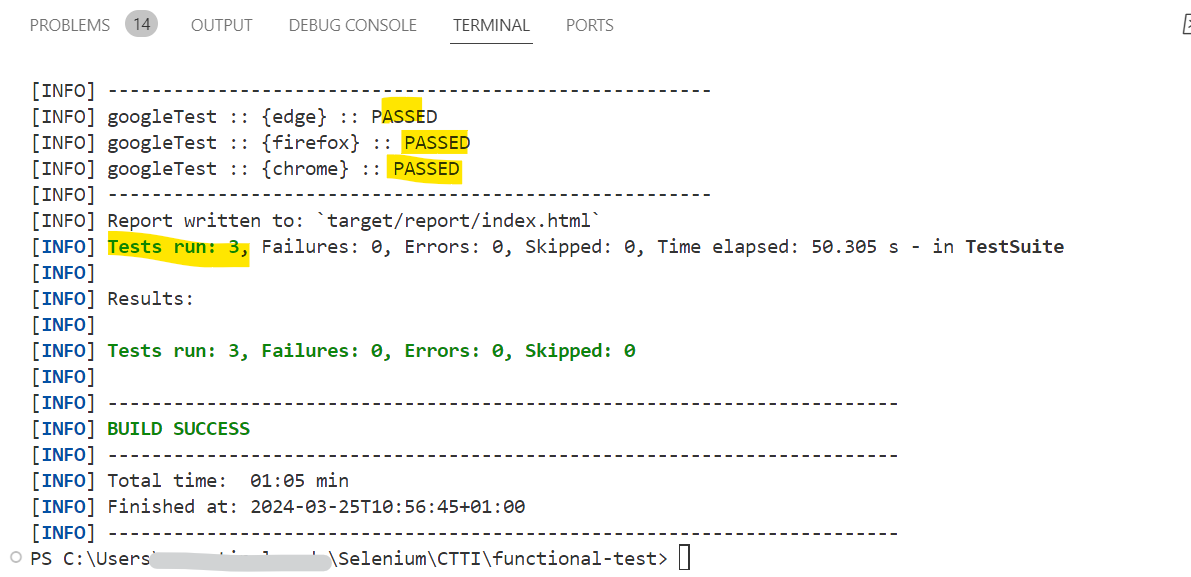
  }

}

#### **Executar i comprobar el report HTML.**

Executar de nou les proves per a comprovar que s'ha solucionat l'error.

*mvn clean test -Dselenium\_url="http://localhost:4444/wd/hub" -Dapp\_url="https://www.google.com"*



En el index.html, ja es veuen totes les proves passades.

