

Tema 12

Testing Frontend

Micael Gallego
micael.gallego@urjc.es
@micael_gallego

Testing frontend

- Introducción a Selenium
- Selenium IDE
- Selenium WebDriver
- Selenium y Jest
- Selenium Grid
- Alternativas a Selenium

Introducción a Selenium

- Las **pruebas de sistema funcionales** de aplicaciones **web** consisten en automatizar las acciones que **realizaría un usuario** usando la **web**
- Existen **tecnologías** que permiten manejar un navegador web de forma automatizada y leer el **contenido** de la página para poder **verificar** que el comportamiento es el esperado

Introducción a Selenium

- La tecnología de **control de navegadores web** más usada es **Selenium**
- Permite manejar **cualquier navegador web** desde **cualquier lenguaje de programación**



Introducción a Selenium

- **Selenium** es un **framework** que permite la **automatización** de **pruebas** para aplicaciones **web**
- Licencia Apache 2.0
- Diseñado inicialmente en 2004 por Jason Huggins
- El nombre fue elegido como burla de la herramienta comercial de pruebas Mercury (actualmente HP Unified Functional Testing)

“Selenium is a key mineral which protects the body from Mercury toxicity”

<http://www.seleniumhq.org/>



Introducción a Selenium

- Selenium tiene tres componentes:

Proyecto	Descripción
Selenium IDE	 Plugin de navegador web que permite grabación y reproducción de navegación en aplicaciones web
Selenium WebDriver	 Control automatizado de navegadores web locales
Selenium Grid	 Control automatizado de navegadores web remotos

Testing frontend

- Introducción a Selenium
- **Selenium IDE**
- Selenium WebDriver
- Selenium y Jest
- Selenium Grid
- Alternativas a Selenium

Introducción a Selenium

- Selenium tiene tres componentes:

Proyecto	Descripción
Selenium IDE	 Plugin de navegador web que permite grabación y reproducción de navegación en aplicaciones web
Selenium WebDriver	 Control automatizado de navegadores web locales
Selenium Grid	 Control automatizado de navegadores web remotos

Selenium IDE

- Es un plugin de Firefox y Chrome que permite **grabar y reproducir** interacciones con aplicaciones web



<https://www.seleniumhq.org/selenium-ide/>

Selenium IDE

Selenium IDE - the-internet*

Project: the-internet*

Executing

login succeeded*

http://the-internet.herokuapp.com

	Command	Target	Value
1	open	/	
2	click	linkText=Form Authentication	
3	type	id=username	tomsmith
4	type	id=password	SuperSecretPassword!
5	send keys	id=password	\${KEY_ENTER}
6	assert element present	id=flash	You logged into a secure area!\n

Command: assert element present

Target: id=flash

Value: id=flash

Description: css=#flash

Opens Window: xpath=/div[@id='flash']

Log Reference

Runs: 1 Failures: 0

10

Testing frontend

- Introducción a Selenium
- Selenium IDE
- **Selenium WebDriver**
- Selenium y Jest
- Selenium Grid
- Alternativas a Selenium

Selenium WebDriver

- Selenium tiene tres componentes:

Proyecto	Descripción
Selenium IDE	 Plugin de navegador web que permite grabación y reproducción de navegación en aplicaciones web
Selenium WebDriver	 Control automatizado de navegadores web locales
Selenium Grid	 Control automatizado de navegadores web remotos

Selenium WebDriver

- Permite manejar un navegador web usando un lenguaje de programación estándar
- Compatibilidad:
 - Navegadores: Chrome, Firefox, Internet Explorer, Opera, Safari, Edge
 - Navegadores móviles: Android, iOS, Windows Phone
 - Navegadores “headless”: HtmlUnit, PhantomJS
 - Sistemas operativos: Windows, Linux, Mac OS X
 - Lenguajes: C#, Haskell, Java, JavaScript, Objective-C, Perl, PHP, Python, R, Ruby
 - <http://docs.seleniumhq.org/projects/webdriver/>



Selenium WebDriver

- Selenium WebDriver es un **protocolo de red** para manejar navegadores web desde código
- Formado por **dos partes**:
 - **Librería del lenguaje (cliente)**: Se usa como cualquier otra librería NPM
 - **Binario específico de cada navegador web (servidor)**:
 - Tiene que descargarse manualmente
 - Se guarda en el PATH del sistema o en la carpeta donde se ejecuta el programa Node

Selenium WebDriver

- **Descarga de binarios:**

- Chrome: <https://sites.google.com/a/chromium.org/chromedriver/>
- Firefox: <https://github.com/mozilla/geckodriver/>
- Opera: <http://choice.opera.com/developer/tools/operadriver/>
- Edge:
<https://www.microsoft.com/en-us/download/details.aspx?id=48212>
- Safari: Hay que instalar manualmente una extensión
- Internet Explorer:
<https://code.google.com/p/selenium/wiki/InternetExplorerDriver> (y además hay que cambiar la configuración de seguridad)

- **Descarga de dependencia NPM**

```
$ npm install --save-dev selenium-webdriver
```

- Normalmente se configura como **devDependency** porque se utiliza para implementar tests
- Se puede añadir como **dependencia normal** si se implementa un *crawler* o *scraper* que descarga información de Internet de forma automática

<https://selenium.dev/selenium/docs/api/javascript/index.html>

Selenium WebDriver: JavaScript

- **Acciones en el test:**

- Crear un objeto WebDriver específico para el navegador que queramos utilizar
- Abrir una página web (URL)
- Localizar elementos (WebElement)
- Interactuar con elementos (hacer click, leer atributos, etc)
- Esperar a que ciertos elementos estén disponibles (carga de la página)
- Verificar que la web bajo pruebas cumple las condiciones esperadas (aserciones)

Selenium WebDriver: JavaScript

ejem1

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

async function browse() {

  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.titleContains('webdriver'), 5000);
    console.log("Google works as expected");

  } catch(e){

    console.log("Google does not show search item in the title");

  } finally {
    await driver.quit();
  }
}

browse();
```

Selenium WebDriver: JavaScript

- Crear el objeto WebDriver

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

- Abrir una página web

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

Selenium WebDriver: JavaScript

- Localizar elementos en la página

```
// Locate single element
var webElement1 = await driver.findElement(By.id("id"));
var webElement2 = await driver.findElement(By.name("name"));
var webElement3 = await driver.findElement(By.className("class"));
var webElement4 = await driver.findElement(By.cssSelector("cssInput"));
var webElement5 = await driver.findElement(By.linkText("text"));
var webElement6 = await driver.findElement(By.partialLinkText("partial text"));
var webElement7 = await driver.findElement(By.tagName("tag name"));
var webElement8 = await driver.findElement(By.xpath("/html/body/div[4]"));

// Locate element list
var webElements = driver.findElements(By...);
```

Selenium WebDriver: JavaScript

- Interactuar con los elementos (hacer click, leer atributos, etc)

```
await webElement1.click();
await webElement1.clear();
await webElement1.sendKeys("text");

var text = await webElement1.getText();
var href = await webElement1.getAttribute("href");
var css = await webElement1.getCssValue("css");
var dim = await webElement1.getSize();

var enabled = await webElement1.isEnabled();
var selected = await webElement1.isSelected();
var displayed = await webElement1.isDisplayed();
```

Selenium WebDriver: JavaScript

- En ocasiones necesitamos **esperar** a que ciertos elementos alcancen un resultado esperado

```
await driver.wait(until.elementToBeClickable(By.id("id1")),1000);
await driver.wait(until.elementToBeSelected(By.id("id2")),1000);
await driver.wait(until.presenceOfElementLocated(By.id("id3")),1000);
await driver.wait(until.titleIs("Page title"),1000);
await driver.wait(
    until.textToBePresentInElementLocated(By.tagName("body"), "text"),1000);
```

- Se pueden usar como aserciones del test si no se obtiene el resultado esperado en el timeout indicado

Testing frontend

- Introducción a Selenium
- Selenium IDE
- Selenium WebDriver
- **Selenium y Jest**
- Selenium Grid
- Alternativas a Selenium

Selenium y Jest

ejem2

- Selenium se puede usar en los tests Jest

web.test.js

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

test('Google shows search term in title', async () => {

  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.titleContains('webdriver'), 5000);

  } finally {
    await driver.quit();
  }
})
```

Selenium y Jest

ejem2

- El driver se puede crear en `beforeEach()`

`web2.test.js`

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

let driver;

beforeEach(async () => {
  driver = await new Builder().forBrowser('chrome').build();
})

afterEach(async () => {
  if (driver) {
    await driver.quit();
  }
});

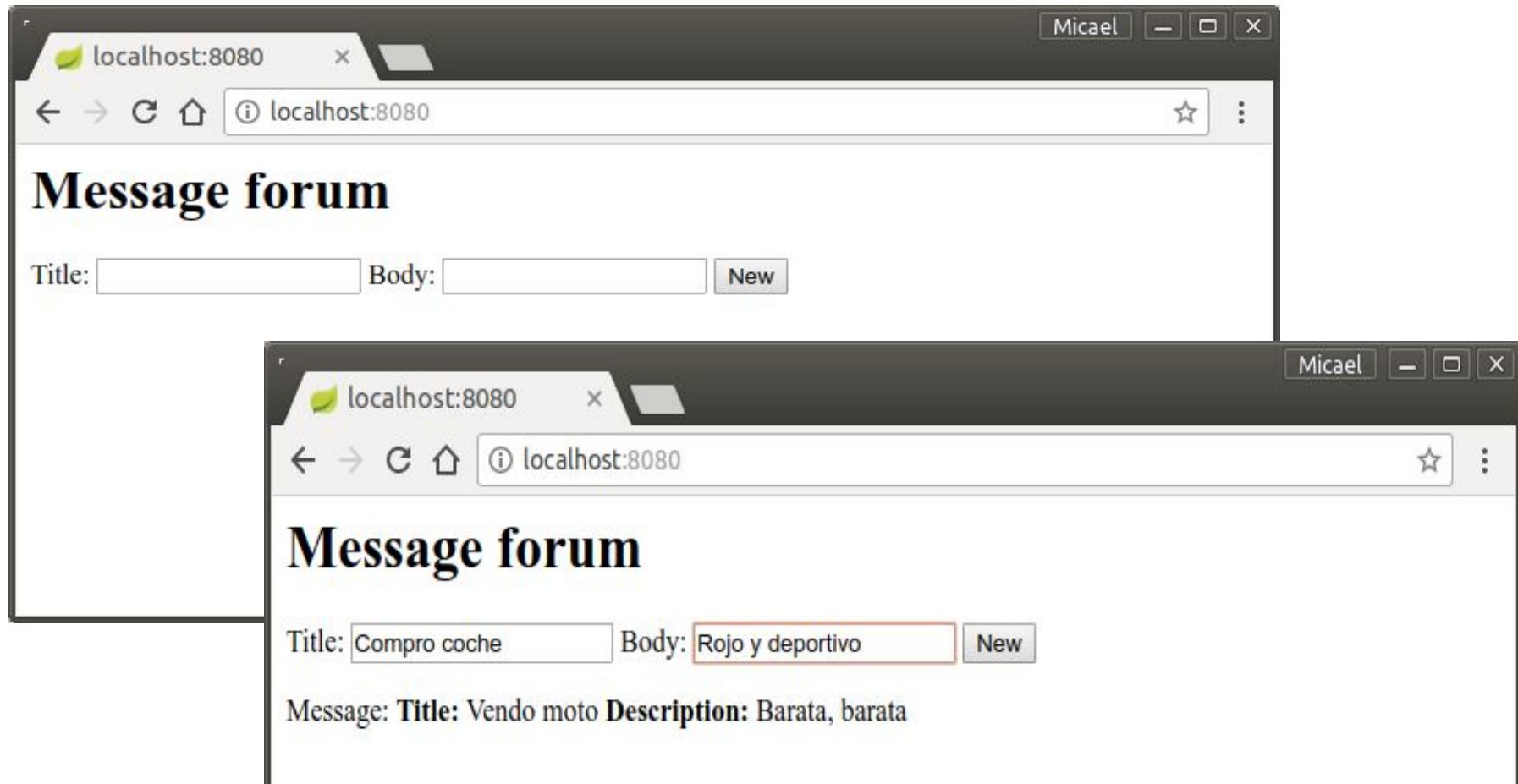
test('Google shows search term in title', async () => {

  await driver.get('http://www.google.com/ncr');
  await driver.findElement(By.name('q')).sendKeys('webdriver', Key.RETURN);
  await driver.wait(until.titleContains('webdriver'), 5000);

})
```

Selenium y Jest

- Ejemplo



Selenium y Jest

```
..<html> == $0
  ▶ <head>...</head>
  ▼ <body>
    <h1>Message forum</h1>
    ▼ <form id="form" action="/" method="post">
      "Title: "
      <input id="title-input" type="text" name="title">
      "Body: "
      <input id="body-input" type="text" name="body">
      <input id="submit" type="submit" value="New">
    </form>
    ▼ <div id="messages">
      ▼ <p>
        "Message: "
        <b>Title:</b>
        <span id="title">Vendo moto</span>
        <b>Description:</b>
        <span id="body">Barata, barata</span>
        <br>
      </p>
    </div>
  </body>
</html>
```

Selenium y Jest

```
test('Ad is created and shown', async () => {

    //Given
    await driver.get("http://localhost:" + this.port + "/");

    //When
    var newTitle = "MessageTitle";
    var newBody = "MessageBody";

    await driver.findElement(By.id("title-input")).sendKeys(newTitle);
    await driver.findElement(By.id("body-input")).sendKeys(newBody);

    await driver.findElement(By.id("submit")).click();

    //Then
    var title = await driver.findElement(By.id("title")).getText();
    var body = await driver.findElement(By.id("body")).getText();

    expect(title).toBe(newTitle);
    expect(body).toBe(newBody);

})
```

Selenium y Jest

```
test('Ad is created and shown', async () => {

    //Given
    await driver.get("http://localhost:" + this.port + "/");

    //When
    var newTitle = "MessageTitle";
    var newBody = "MessageBody";

    await driver.findElement(By.id("title-input")).sendKeys(newTitle);
    await driver.findElement(By.id("body-input")).sendKeys(newBody);

    await driver.findElement(By.id("submit")).click();

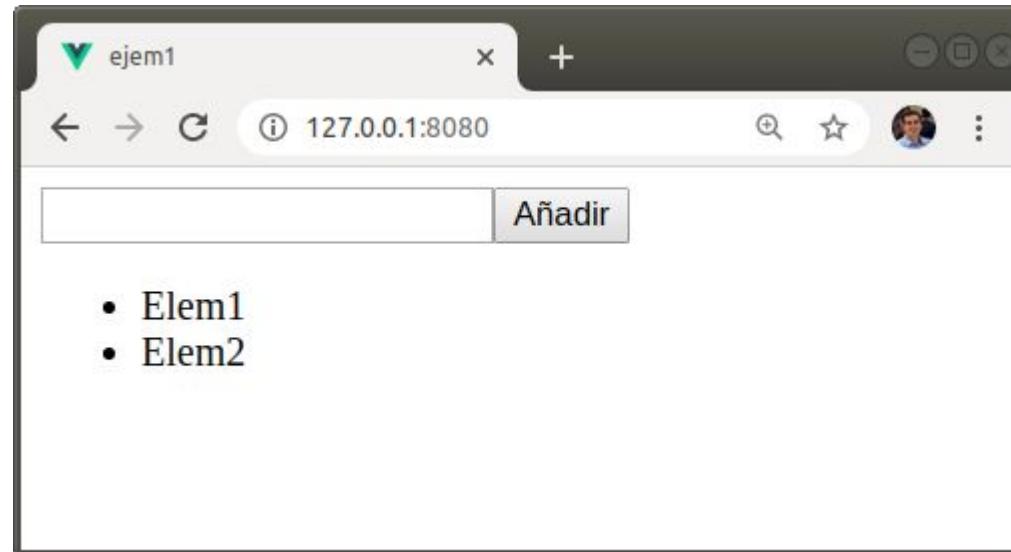
    //Then
    var title = await driver.findElement(By.id("title")).getText();
    var body = await driver.findElement(By.id("body")).getText();

    expect(title).toBe(newTitle);
    expect(body).toBe(newBody);

})
```

Ejercicio 1

- Implementa un test de una aplicación que añade a la página el insertado en un campo de un formulario



Testing frontend

- Introducción a Selenium
- Selenium IDE
- Selenium WebDriver
- Selenium y Jest
- **Selenium Grid**
- Alternativas a Selenium

Selenium Grid

- Selenium tiene tres componentes:

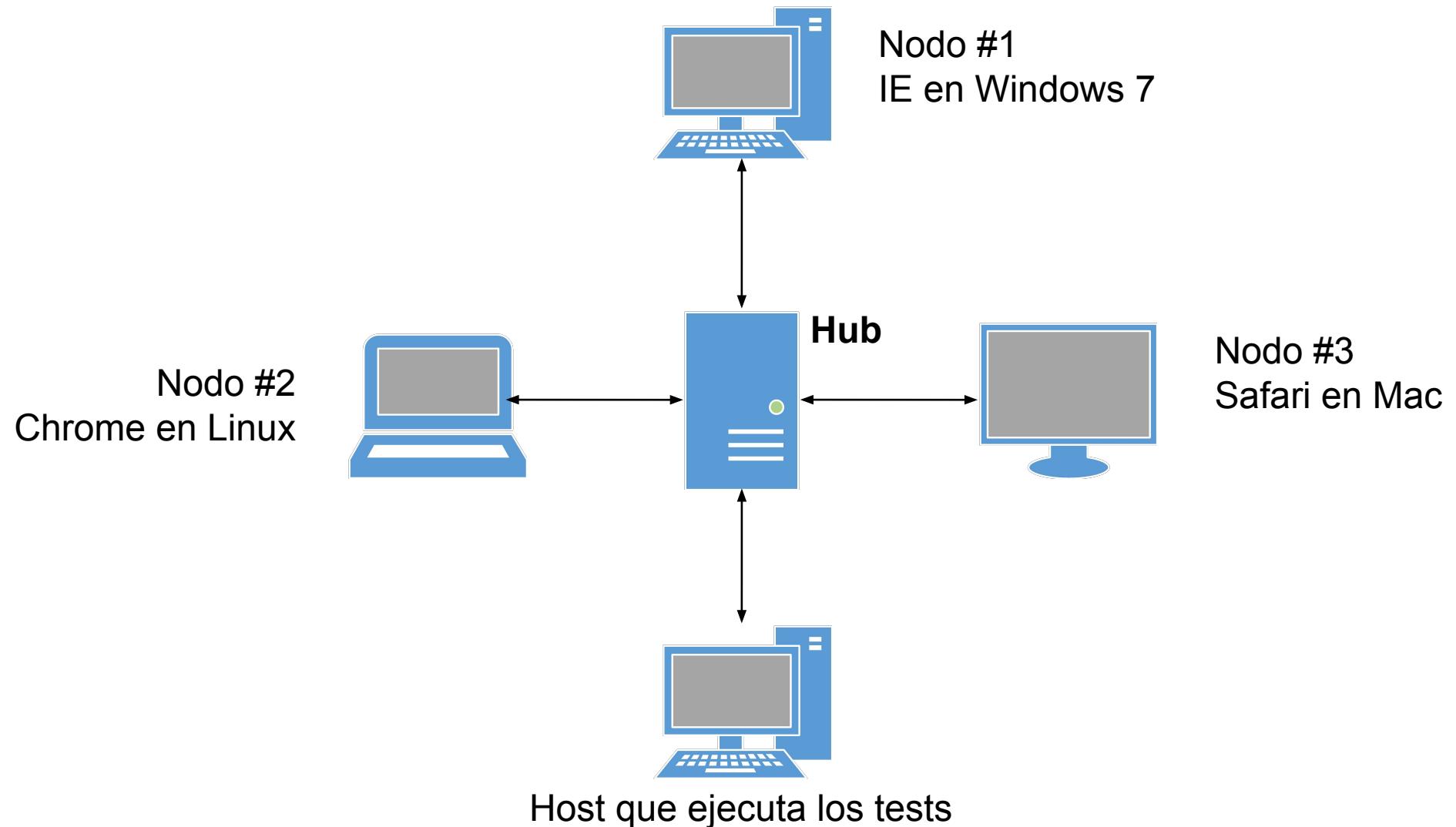
Proyecto	Descripción
Selenium IDE	 Plugin de navegador web que permite grabación y reproducción de navegación en aplicaciones web
Selenium WebDriver	 Control automatizado de navegadores web locales
Selenium Grid	 Control automatizado de navegadores web remotos

Selenium Grid

- Con **Selenium WebDriver** usamos los navegadores instalados de forma local en la máquina que está ejecutando los tests
- **Selenium Grid** permite el control de navegadores instalados en otras máquinas (en remoto)
- **Arquitectura Selenium Grid:**
 - **Hub (Maestro):** Pieza central de la infraestructura que orquesta la ejecución de la prueba
 - **Nodos:** Máquinas que aportan navegadores en los que ejecutar pruebas



Selenium Grid



Selenium Grid

- **Hub**

```
java -jar selenium-server-standalone-3.8.1.jar -role hub -port 4444
```

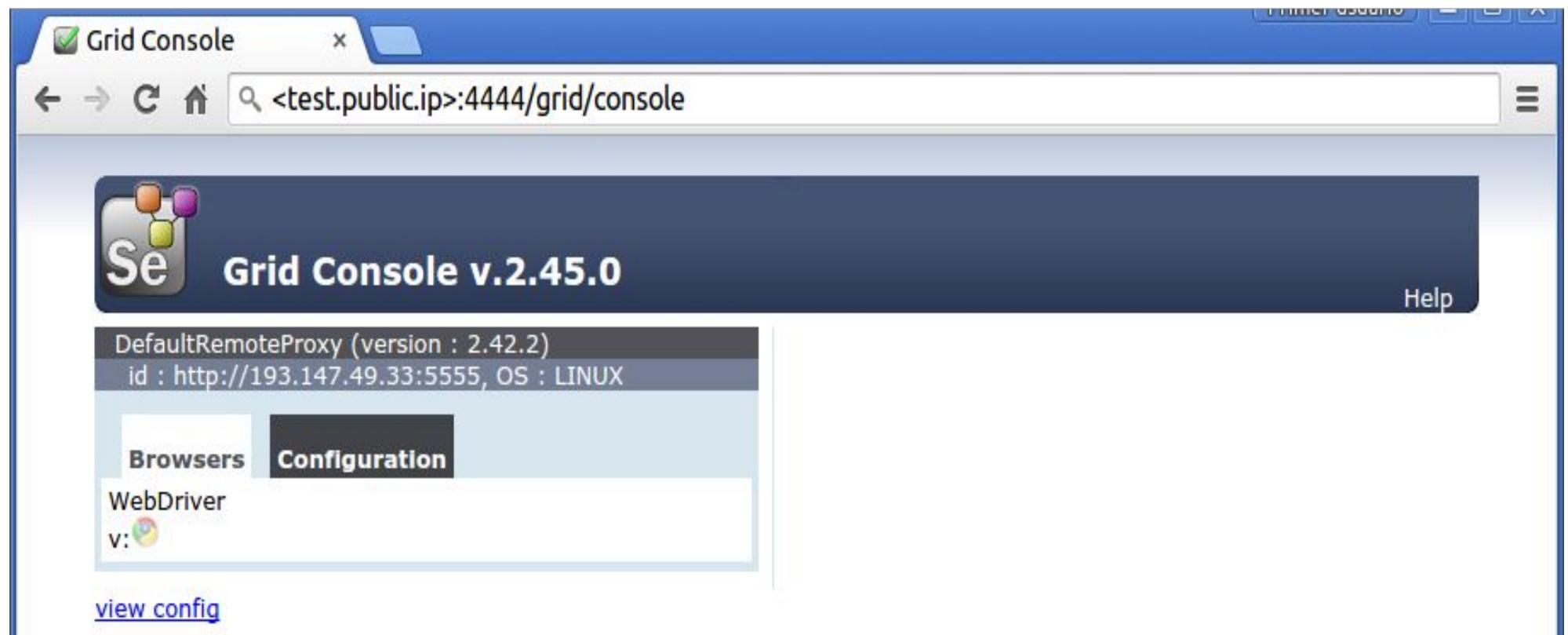
- **Nodos**

```
java -jar selenium-server-standalone-3.8.1.jar -role node -port  
<node-port> -hub http://<hub-address>:<hub-port>/grid/register -browser  
browserName=<browser-name>, version=<browser-version>,  
maxInstances=<max-instances>, platform=<platform> -maxSession  
<max-sessions> -Dwebdriver.chrome.driver=${remoteChromeDriver} -timeout  
<seconds>
```

<http://www.seleniumhq.org/download/>

Selenium Grid

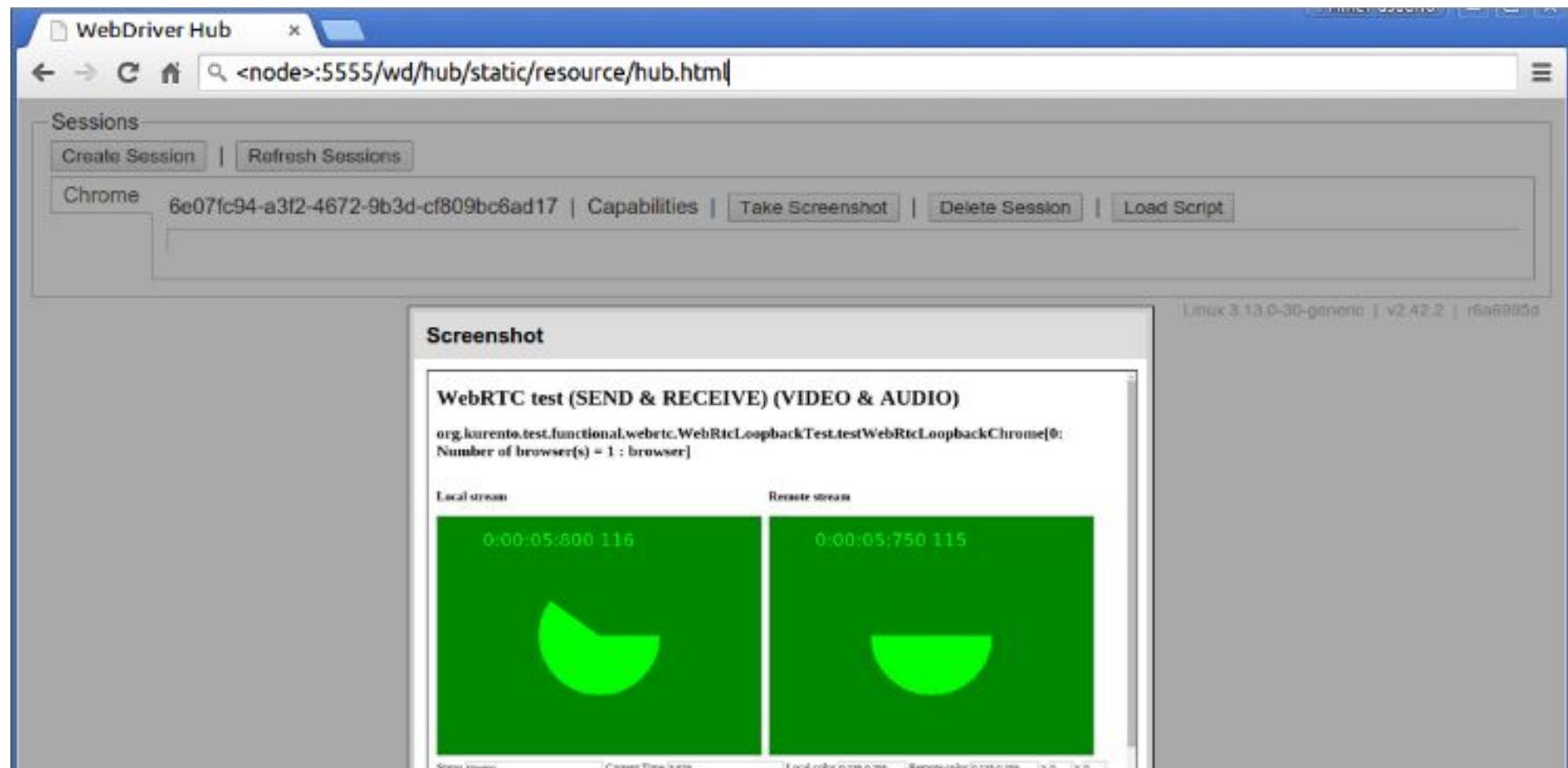
- Consola de administración del Hub



<http://<hub-address>:<hub-port>/grid/console>

Selenium Grid

- Consola de administración del nodo



<http://<node-address>:<node-port>/wd/hub/static/resource/hub.html>

Selenium Grid

- **Configuración del servidor Selenium Grid:**

- A) Con variables de entorno

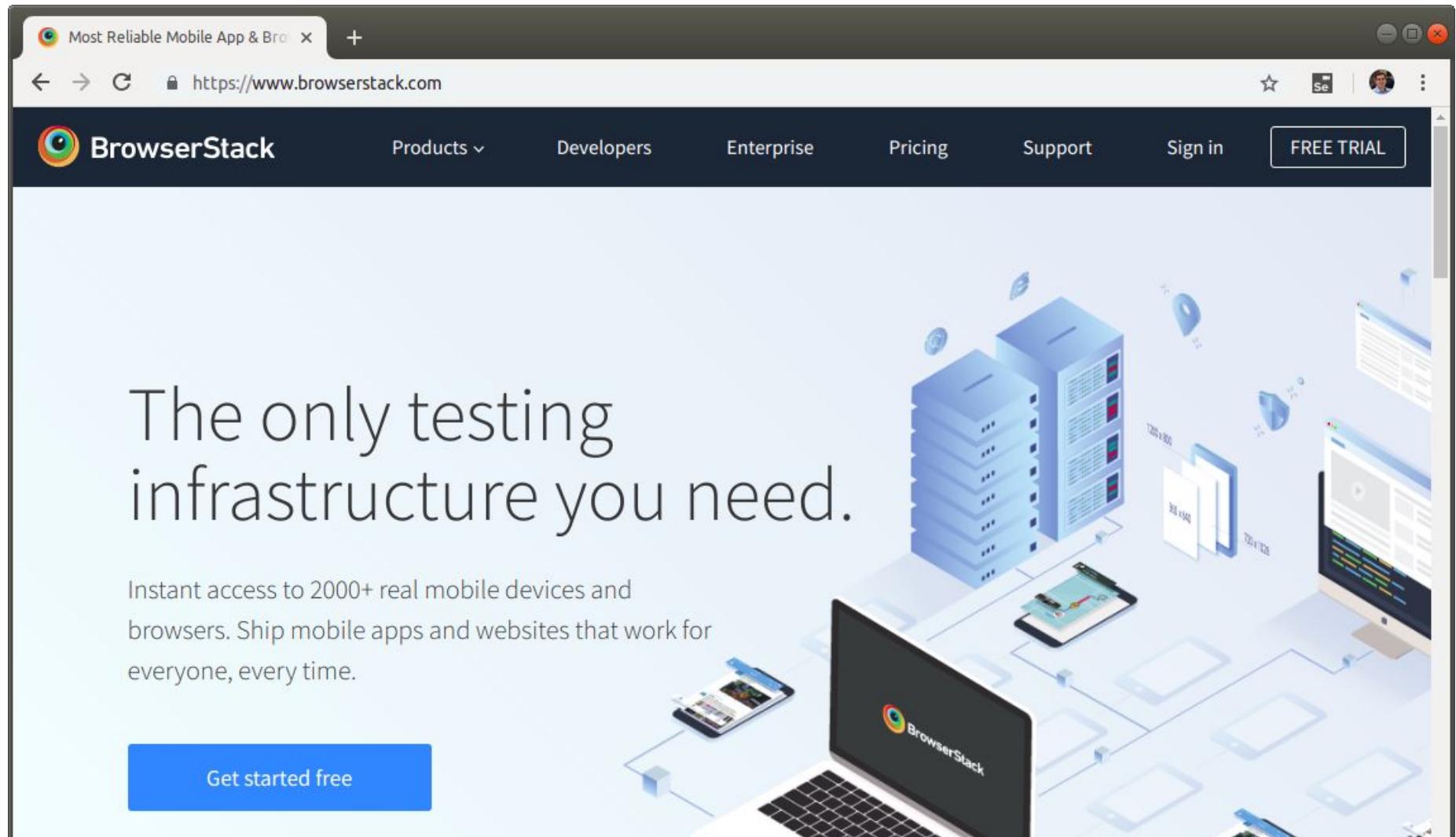
```
SELENIUM_BROWSER=chrome:36:LINUX \
SELENIUM_REMOTE_URL=http://www.example.com:4444/wd/hub \
node mytest.js
```

- B) Con código

```
var driver = new webdriver.Builder()
  .withCapabilities({'browserName': 'chrome', 'idleTimeout':'60'})
  .usingServer('http://localhost:4444/wd/hub')
  .build();
```

Selenium Grid

- Servicios online que ofrecen navegadores web

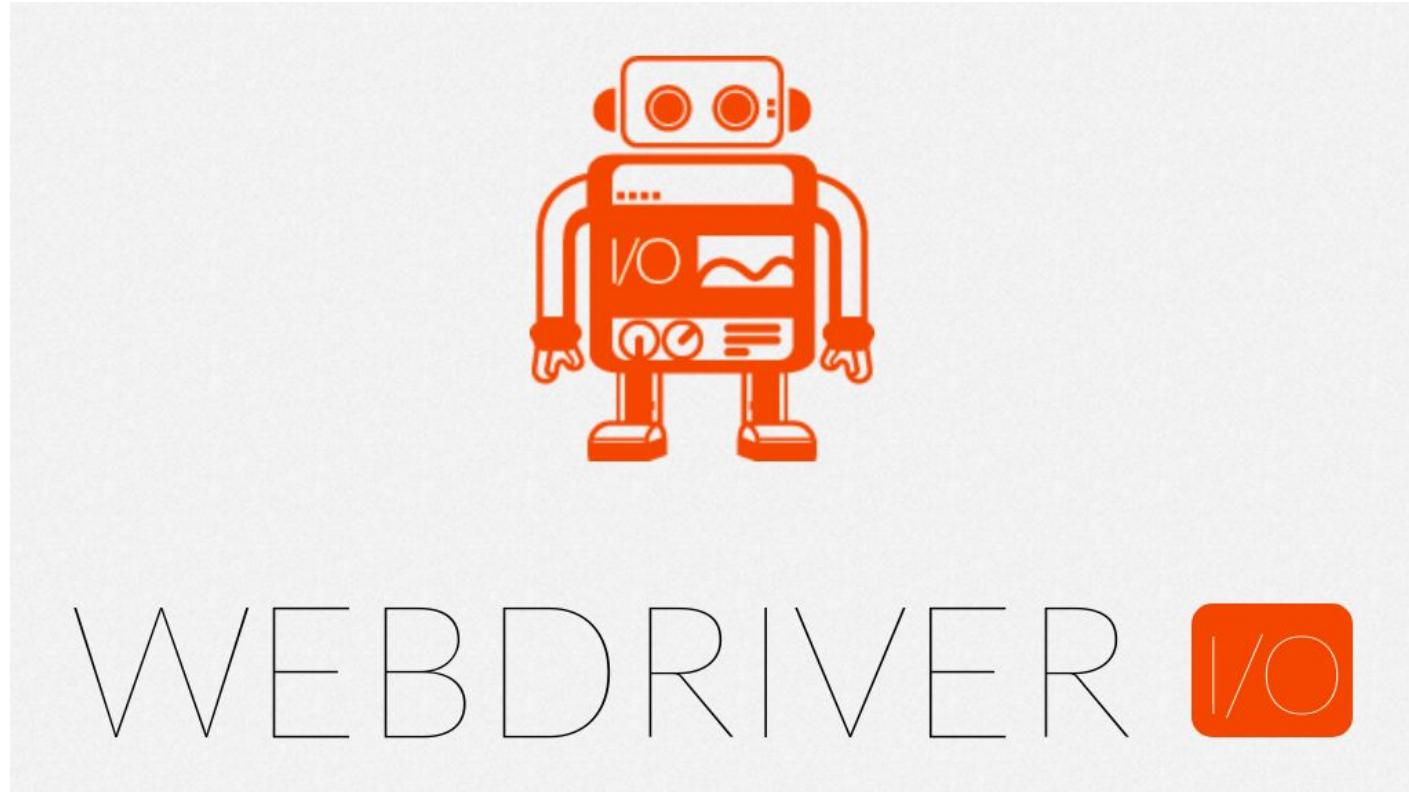


Testing frontend

- Introducción a Selenium
- Selenium IDE
- Selenium WebDriver
- Selenium y Jest
- Selenium Grid
- Alternativas a Selenium

Alternativas a Selenium

- WebDriver.io



<https://webdriver.io/>

Alternativas a Selenium

• WebDriver.io

- Usa el mismo protocolo (y drivers) que Selenium
- La API es más sencilla
- Se puede usar sin async/await
- No se puede* usar con Jest
- Tiene su propio ejecutor de tests y assertions

Alternativas a Selenium

• WebDriver.io

```
const assert = require('assert')

describe('webdriver.io page', () => {

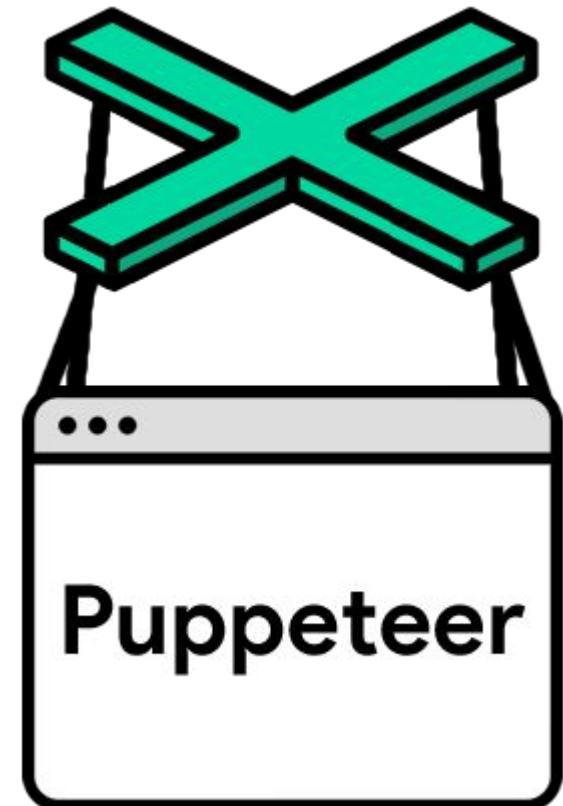
  it('should have the right title', () => {

    browser.url('https://webdriver.io')
    const title = browser.getTitle()
    assert.strictEqual(title, 'WebdriverIO')
  })
})
```

Alternativas a Selenium

• Puppeteer

- Sólo funciona en **Google Chrome**
- Protocolo de comunicación con **más control** del que ofrece Selenium (p.e. impresión)
- Integración nativa con **Jest**



<https://pptr.dev/>

Alternativas a Selenium

• Puppeteer

```
describe('Google', () => {  
  
  it('should be titled "Google"', async () => {  
  
    await page.goto('https://google.com');  
    await expect(page.title()).resolves.toMatch('Google');  
  
  });  
});
```

<https://jestjs.io/docs/en/puppeteer>

Alternativas a Selenium

- Cypress.io



<https://cypress.io/>

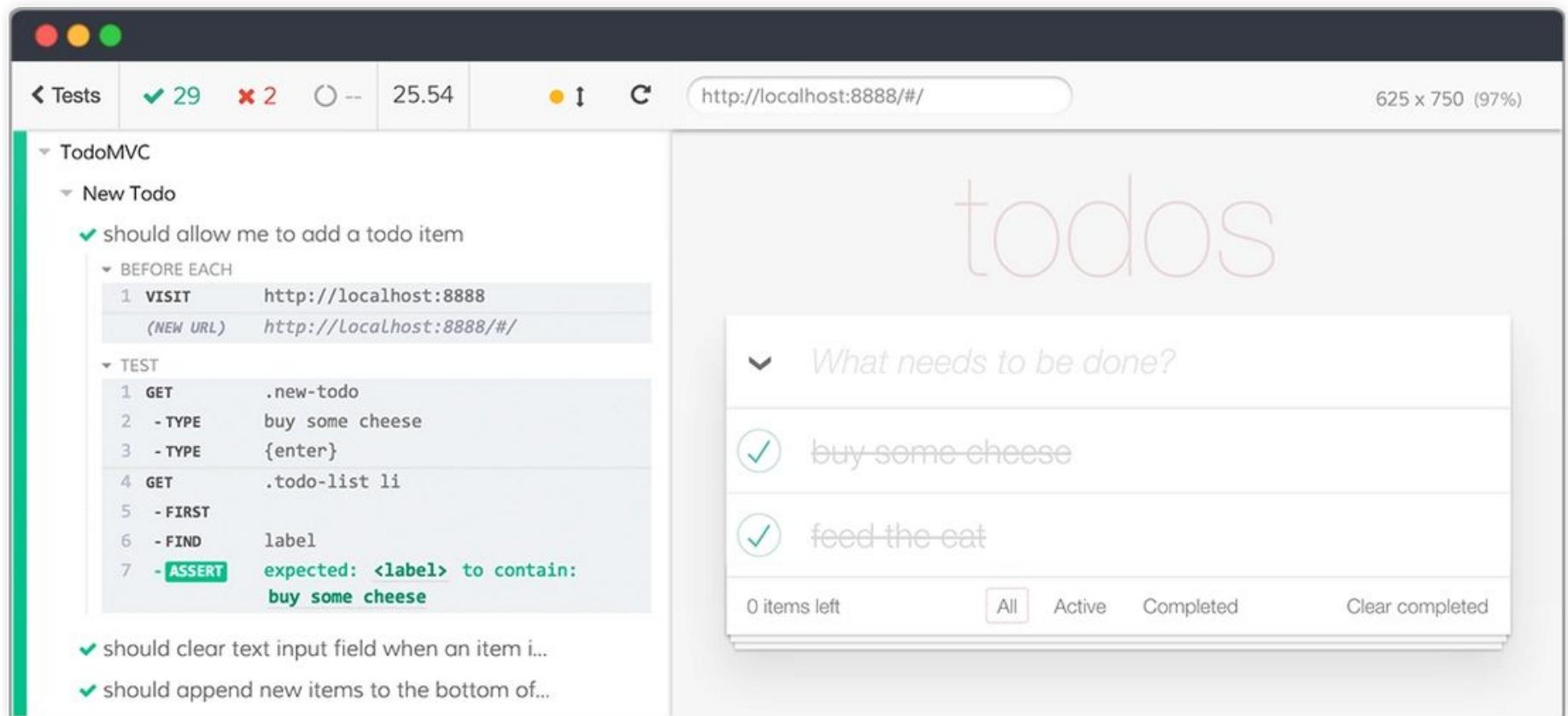
Alternativas a Selenium

- **Cypress.io**

- Sólo funciona en **Google Chrome**
- Ejecuta los test **dentro del propio browser** (no hay protocolo cliente-servidor)
- Eso hace que sean mucho **más rápidos** de ejecutar
- Se tiene mucho **más control** sobre el browser (porque se ejecuta en el browser)

Alternativas a Selenium

- Cypress.io



The screenshot shows the Cypress.io test runner interface. On the left, the test tree for a "TodoMVC" application is displayed, specifically under the "New Todo" section. A test named "should allow me to add a todo item" is expanded, showing the following steps:

```

    1 VISIT      http://localhost:8888
    (NEW URL)   http://localhost:8888/#/
    2 - TYPE     buy some cheese
    3 - TYPE     {enter}
    4 GET        .todo-list li
    5 - FIRST
    6 - FIND     label
    7 - ASSERT   expected: <label> to contain:
                  buy some cheese
  
```

The test has passed (green checkmark). The right side of the interface shows a screenshot of a browser window displaying a "todos" application. The page shows a list of items: "buy some cheese" and "feed the cat", both of which are checked off. At the bottom, there are buttons for "All", "Active", "Completed", and "Clear completed".

Alternativas a Selenium

• Cypress.io

```
describe('My First Test', function() {  
  
  it('clicking "type" navigates to a new url', function() {  
  
    cy.visit('https://example.cypress.io')  
    cy.contains('type').click()  
  
    // Should be on a new URL which includes '/commands/actions'  
    cy.url().should('include', '/commands/actions')  
  })  
})
```