CHOUCAIR
Business Centric Testing

WORKSHOP INTERMEDIO

AUTOMATIZACIÓN CON SELENIUM

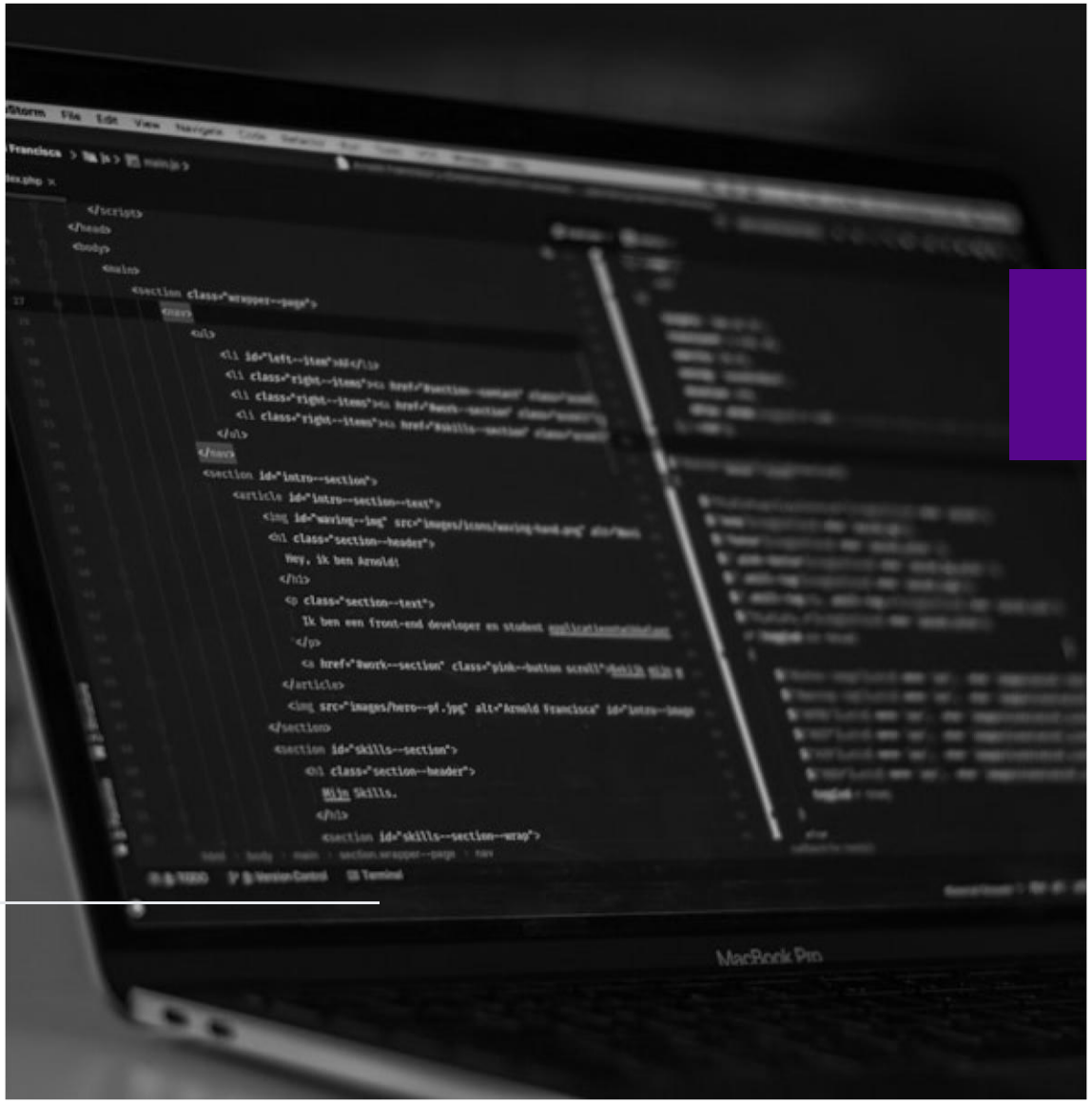
Sesión 03.1

Locators

Estrategias de Localización

Selenium WebDriver

Java • Selenium • Gradle



CHOUCAIR
Business Centric Testing

Índice de Contenidos

¿Qué son los Locators?

Definición

Los Locators son estrategias que utiliza Selenium WebDriver para identificar y localizar elementos en una página web durante la automatización de pruebas.

Conceptos Clave

- Identificación de elementos: Permiten encontrar elementos HTML específicos como botones, campos de texto, enlaces, etc.
- Interacción automatizada: Son fundamentales para que Selenium pueda interactuar con los elementos de la página (hacer clic, escribir texto, leer valores).
- Múltiples estrategias: Selenium ofrece 8 tipos diferentes de locators para adaptarse a distintas situaciones y estructuras HTML.
- Base de la automatización: Sin locators efectivos, no es posible crear scripts de prueba automatizados confiables.

CHOUCAIR
Business Centric Testing

Tipos de Locators

1 ID	2 Name	3 ClassName	4 TagName
5 LinkText	6 PartialLinkText	7 CSS Selector	8 XPath

Selenium WebDriver ofrece 8 estrategias diferentes para localizar elementos en páginas web

ID Locator

Descripción

Localiza elementos en la página web utilizando el atributo id del elemento HTML.

Sintaxis

```
driver.findElement(By.id("elementId"))
```

Ventajas

- Rápido: El método más eficiente de localización
- Único: Los IDs deben ser únicos en la página
- Confiable: Primera opción recomendada cuando está disponible

Ejemplo

```
// Ejemplo en Java
WebElement element = driver.findElement(
  By.id("username")
);
element.sendKeys("testuser");
```

Name Locator

- Descripción: Localiza elementos por su atributo `name`
- Sintaxis:

```
driver.findElement(By.name("elementName"))
```

- Uso común: Muy utilizado en formularios HTML donde los campos tienen atributos `name` definidos
- Ventaja: Ampliamente usado en formularios web, fácil de identificar en el código HTML

```
// HTML Element: <input type="text" name="username" />
WebElement usernameField = driver.findElement(By.name("username"));
usernameField.sendKeys("TestUser");
```

ClassName Locator

Descripción

Localiza elementos HTML mediante su atributo class. Es útil cuando múltiples elementos comparten el mismo estilo CSS.

Sintaxis

```
driver.findElement(By.className("className"))
```

Características

- Puede retornar múltiples elementos si varios comparten la misma clase
- Común en formularios y elementos con estilos CSS
- Solo acepta un nombre de clase (sin espacios)

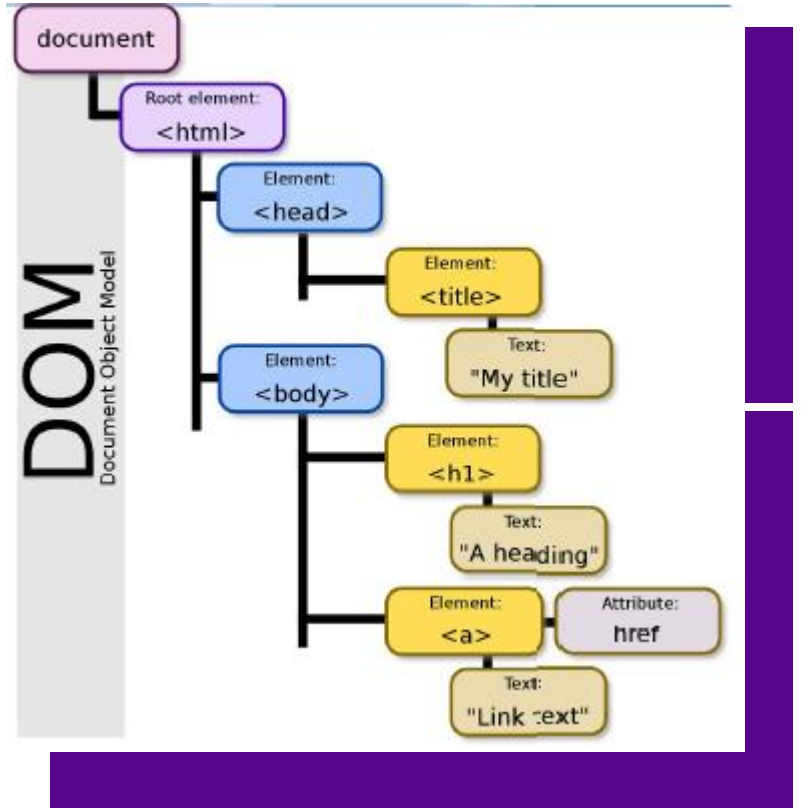
Ejemplo de Código

```
// HTML:
<input class="form-control" />

// Java + Selenium:
WebElement element = driver
  .findElement(By.className("form-control"));
```

TagName Locator

- Descripción: Localiza elementos por su etiqueta HTML (tag)
- Sintaxis: `driver.findElement(By.tagName("tagName"))`
- Uso común: Útil para localizar elementos genéricos como inputs, buttons, divs, etc.
- Ventaja: Simple y directo cuando se trabaja con elementos del mismo tipo
- Nota: Puede retornar múltiples elementos si hay varios con la misma etiqueta



LinkText Locator

Descripción

El LinkText Locator localiza elementos de tipo enlace (<a>) mediante la coincidencia exacta del texto visible del enlace.

Sintaxis

```
WebElement element = driver.findElement(By.linkText("texto del enlace"));
```

Características

- Coincidencia exacta del texto
- Solo para elementos <a>
- Sensible a mayúsculas/minúsculas

Ejemplo HTML

```
<a href="/login">Iniciar Sesión</a>
<a href="/register">Registrarse</a>
```

```
// Localizar el enlace "Iniciar Sesión"
driver.findElement(By.linkText("Iniciar Sesión"));
```

PartialLinkText Locator

- Descripción: Localiza enlaces por texto parcial del enlace
- Sintaxis:

```
driver.findElement(By.partialLinkText("texto parcial"))
```

- Flexibilidad: Más flexible que LinkText, no requiere coincidencia exacta del texto completo
- Ventaja: Útil cuando el texto del enlace es muy largo o puede variar ligeramente
- Uso: Específico para elementos `<a>` (enlaces)

Ejemplo de Código

HTML:

```
<a href="/contact">Contáctanos para más información</a>
```

Selenium (texto completo):

```
driver.findElement(By.linkText("Contáctanos para más información"));
```

Selenium (texto parcial):

```
driver.findElement(By.partialLinkText("Contáctanos"));
```

CHOUCAIR
Business Centric Testing

CSS Selector - Introducción

¿Qué es CSS Selector?

CSS Selector es un locator potente y flexible que utiliza la sintaxis de selectores CSS para identificar elementos en una página web.

Sintaxis:

```
driver.findElement(By.cssSelector("selector"))
```

Ventajas

- Rápido: Excelente rendimiento en la ejecución
- Versátil: Permite combinaciones complejas de selectores
- Familiar: Usa la misma sintaxis que CSS estándar

Ejemplos de Referencia

css = “input # email”

css = “input.inputtext”

css = “input [name = lastName]”

css = “input.inputtext [tabindex = 1]”

CSS Selector - Sintaxis Básica

- Por ID:

```
#idValue
```

Selecciona el elemento con id="idValue"

- Por Clase:

```
.className
```

Selecciona elementos con class="className"

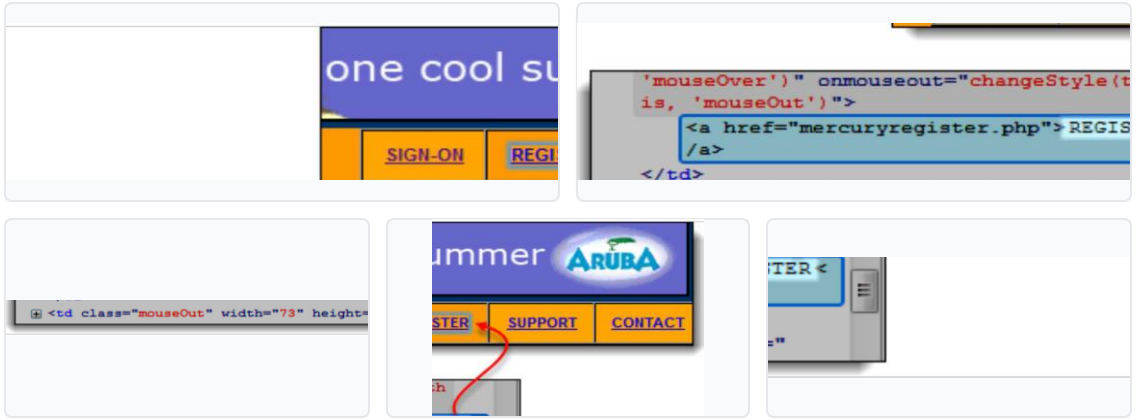
- Por Atributo:

```
[attribute='value']
```

Selecciona elementos con el atributo especificado

- Combinaciones:

```
div.className - div con clase
input[type='text'] - input tipo texto
#id .class - clase dentro de id
```



XPath - Introducción

¿Qué es XPath?

XPath (XML Path Language) es un lenguaje de consulta que permite navegar a través de elementos y atributos en documentos XML y HTML.

Sintaxis en Selenium

```
driver.findElement(By.xpath("xpath"))
```

Dos Tipos de XPath

- XPath Absoluto: Ruta completa desde el nodo raíz
- XPath Relativo: Busca elementos en cualquier nivel del documento

XPath: Absoluto vs Relativo

XPath Absoluto

Características:

- Ruta completa desde la raíz del documento
- Comienza con barra simple `/`
- Especifica cada nivel jerárquico

Ejemplo:

```
/html/body/div/form/input
```

Desventajas:

- Frágil ante cambios en la estructura HTML
- Difícil de mantener
- No recomendado para producción

XPath Relativo

Características:

- Busca en cualquier nivel del documento
- Comienza con doble barra `//`
- Usa atributos para identificar elementos

Ejemplo:

```
//input[@id='username']
```

Ventajas:

- Más robusto ante cambios estructurales
- Más fácil de escribir y mantener
- Recomendado para automatización

XPath - Funciones Comunes

1 contains()

Busca elementos que contengan un valor parcial en un atributo o texto

```
//div[contains(@class,'error')]
```

2 starts-with()

Localiza elementos cuyo atributo comienza con un valor específico

```
//input[starts-with(@id,'user')]
```

3 text()

Selecciona elementos por su contenido de texto exacto

```
//button[text()='Submit']
```

4 Operadores Lógicos

Combina múltiples condiciones con and / or

```
//input[@type='text' and @name='username']
```

```
//button[@type='submit' or @class='btn-primary']
```

XPath: Ejes de Navegación

Los ejes de XPath permiten navegar por la estructura del DOM en diferentes direcciones desde un nodo de contexto.

- **parent::**

  Selecciona el elemento padre del nodo actual

  ```
  //input[@id='username']/parent::div
  ```

- **following-sibling::**

  Selecciona hermanos siguientes al mismo nivel

  ```
  //label[@for='email']/following-sibling::input
  ```

- **ancestor::**

  Selecciona todos los ancestros (padres, abuelos, etc.)

  ```
  //input[@id='password']/ancestor::form
  ```

- **child::**

  Selecciona todos los elementos hijos directos

  ```
  //div[@class='form']/child::input
  ```

- **preceding-sibling::**

  Selecciona hermanos anteriores al mismo nivel

  ```
  //button[@type='submit']/preceding-sibling::input
  ```

- **descendant::**

  Selecciona todos los descendientes (hijos, nietos, etc.)

  ```
  //form[@id='login']/descendant::input
  ```

> Nota: Los ejes son especialmente útiles cuando no hay atributos únicos disponibles y necesitas navegar por la estructura del DOM de forma relativa.

Mejores Prácticas

1

Preferir ID cuando esté disponible

El locator por ID es el más rápido y confiable. Si un elemento tiene un ID único y estable, úsalo como primera opción para garantizar rendimiento y precisión.

2

Usar CSS Selector para rapidez

Los selectores CSS son más rápidos que XPath y ofrecen gran flexibilidad. Son ideales para la mayoría de los casos donde ID no está disponible.

3

XPath para navegación compleja

Reserva XPath para situaciones que requieren navegación compleja en el DOM, como buscar elementos padre o usar ejes. Prefiere XPath relativo sobre absoluto.

4

Evitar XPath absoluto

Los XPath absolutos son frágiles y se rompen fácilmente con cambios en la estructura HTML. Siempre usa XPath relativo para mayor robustez y mantenibilidad.

5

Mantener locators simples y legibles

Escribe locators claros y fáciles de entender. Evita expresiones excesivamente complejas que dificulten el mantenimiento y la depuración del código de pruebas.

Comparación de Locators

Locator	Velocidad	Flexibilidad	Mantenibilidad
ID	⭐⭐⭐⭐⭐	⭐⭐	⭐⭐⭐⭐⭐
Name	⭐⭐⭐⭐	⭐⭐⭐	⭐⭐⭐⭐
ClassName	⭐⭐⭐⭐	⭐⭐⭐	⭐⭐⭐
TagName	⭐⭐⭐⭐	⭐⭐	⭐⭐
LinkText	⭐⭐⭐⭐	⭐⭐	⭐⭐⭐
PartialLinkText	⭐⭐⭐	⭐⭐⭐	⭐⭐⭐
CSS Selector	⭐⭐⭐⭐⭐	⭐⭐⭐⭐	⭐⭐⭐⭐
XPath	⭐⭐⭐	⭐⭐⭐⭐⭐	⭐⭐⭐

ID: Primera opción cuando está disponible
CSS Selector: Mejor balance velocidad/flexibilidad
XPath: Para navegación compleja en el DOM

Referencia visual de locators en acción

Estrategias de Selección

Jerarquía de Preferencia

- 1. ID - Primera opción cuando está disponible. Único, rápido y confiable.
- 2. Name - Ideal para formularios. Común y fácil de mantener.
- 3. CSS Selector - Flexible y rápido. Buena opción para elementos complejos.
- 4. XPath Relativo - Para navegación compleja y relaciones entre elementos.
- 5. LinkText/PartialLinkText - Específico para enlaces de texto.
- 6. ClassName/TagName - Último recurso. Puede retornar múltiples elementos.

> 💡 Evitar: XPath absoluto - frágil ante cambios en la estructura HTML

Herramientas de Desarrollo

Las herramientas de desarrollo del navegador son esenciales para identificar y probar locators de manera efectiva.

Herramientas Principales

- Chrome DevTools: Herramienta integrada para inspeccionar elementos HTML, ver atributos y estructura del DOM.
- Consola del navegador: Permite probar selectores CSS y XPath en tiempo real usando `$('selector')` y `$x('xpath')`.
- ChroPath: Extensión de Chrome que genera automáticamente XPath y CSS Selectors optimizados.
- Selenium IDE: Herramienta de grabación y reproducción que ayuda a identificar locators durante la interacción con la página.

Manejo de Elementos Dinámicos

- Esperas Implícitas: Configuran un tiempo de espera global para todos los elementos

```
driver.manage().timeouts()
  .implicitlyWait(10, TimeUnit.SECONDS);
```

- Esperas Explícitas: Esperan condiciones específicas para elementos particulares

```
WebDriverWait wait = new WebDriverWait(driver, 10);
wait.until(ExpectedConditions.visibilityOf(element));
```

- IDs Dinámicos: Usar atributos estables como data-attributes, clases CSS consistentes, o XPath con funciones como contains() y starts-with()

- Atributos Estables: Preferir name, data-testid, aria-label sobre IDs generados dinámicamente

Errores Comunes y Soluciones

NoSuchElementException

Causa: El elemento no se encuentra en el DOM o el locator es incorrecto.

Solución:

- Verificar que el locator sea correcto
- Usar esperas explícitas (WebDriverWait)
- Confirmar que el elemento esté visible

StaleElementReferenceException

Causa: El elemento fue localizado pero el DOM cambió y la referencia quedó obsoleta.

Solución:

- Re-localizar el elemento antes de usarlo
- Evitar guardar referencias de elementos
- Usar try-catch para re-intentar

TimeoutException

Causa: El elemento no apareció en el tiempo especificado por la espera explícita.

Solución:

- Aumentar el tiempo de espera
- Verificar condiciones de espera correctas
- Revisar si hay errores de carga en la página

ElementNotInteractableException

Causa: El elemento existe pero no es interactuable (oculto, deshabilitado, cubierto).

Solución:

- Esperar a que el elemento sea clickeable
- Hacer scroll hasta el elemento
- Usar JavaScript Executor como alternativa

Consejo: Implementar manejo de excepciones robusto y logs detallados ayuda a identificar y resolver estos errores rápidamente en

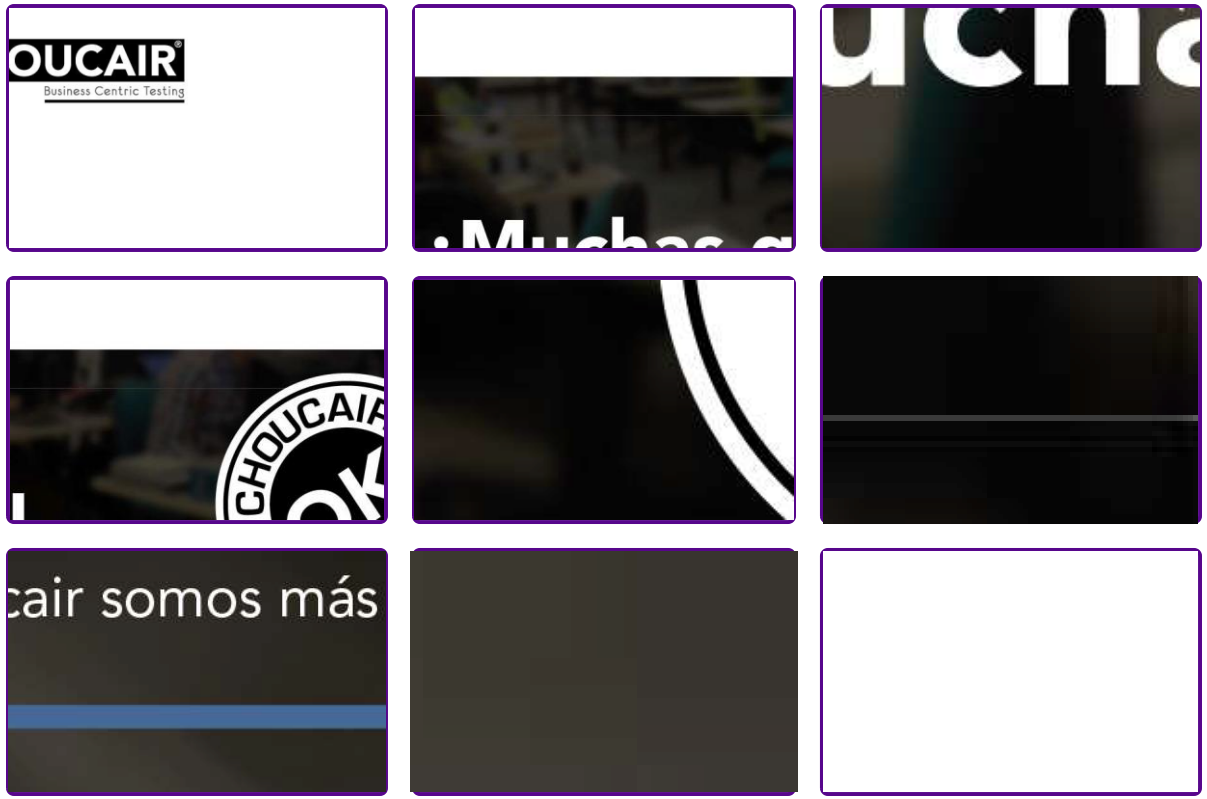
Ejercicio Práctico

Tareas a Realizar

- Identificar elementos usando ID locator
- Localizar campos de formulario con Name locator
- Usar CSS Selector para elementos complejos
- Implementar XPath para navegación en el DOM
- Localizar enlaces con LinkText y PartialLinkText
- Comparar eficiencia de diferentes locators

> Objetivo: Practicar todos los tipos de locators aprendidos en una página web real

Página de Ejemplo



Recursos Adicionales

Documentación Oficial

- Selenium WebDriver Docs: selenium.dev/documentation
- Locating Elements Guide: Guía completa de estrategias de localización
- API Reference: Referencia detallada de métodos y clases

Tutoriales Recomendados

- Test Automation University: Cursos gratuitos de Applitools
- Guru99 Selenium Tutorial: Guías paso a paso con ejemplos
- Selenium Easy: Ejercicios prácticos y desafíos
- YouTube Channels: Automation Step by Step, Software Testing Help

Comunidades y Foros

- Stack Overflow: Tag [selenium] para preguntas técnicas
- Selenium Users Group: Grupo oficial en Google Groups
- Reddit r/selenium: Comunidad activa con tips y soluciones
- GitHub Discussions: Repositorio oficial de Selenium

Herramientas Complementarias

- ChroPath: Extensión de Chrome para generar XPath y CSS
- Selenium IDE: Grabador de pruebas para navegadores
- Chrome DevTools: Inspector de elementos integrado
- Katalon Recorder: Alternativa a Selenium IDE
- SelectorGadget: Herramienta para identificar selectores CSS

Resumen

Conceptos Fundamentales

- Los Locators son estrategias para identificar elementos web
- Fundamentales para la automatización con Selenium
- Permiten interactuar con elementos HTML de forma programática

8 Tipos de Locators

- ID
- ClassName
- LinkText
- CSS Selector
- Name
- TagName
- PartialLinkText
- XPath

Comparación de Velocidad

- Más rápido: ID, Name
- Rápido: CSS Selector
- Más lento: XPath

Mejores Prácticas

- Preferir ID cuando esté disponible (único y rápido)
- Usar CSS Selector para rapidez y flexibilidad
- Aplicar XPath para navegación compleja del DOM
- Evitar XPath absoluto (frágil ante cambios)
- Mantener locators simples y legibles
- Usar atributos estables para elementos dinámicos

Herramientas Útiles

- Chrome DevTools para inspeccionar elementos
- Extensiones: ChroPath, Selenium IDE
- Consola del navegador para probar selectores

Próximos Pasos

- Practicar con diferentes tipos de locators
- Implementar estrategias de espera
- Crear scripts de automatización robustos

Gracias

¿Preguntas?

WS Intermedio Sesión 03.1

Locators - Selenium WebDriver