



2.3 – Pruebas de Servicios de Internet

# Tema 4 – Pruebas de UI en Java (Selenium)



Universidad  
Rey Juan Carlos

Micael Gallego  
micael.gallego@urjc.es  
@micael\_gallego

Michel Maes  
michel.maes@urjc.es



- Introducción a Selenium
- Selenium IDE
- Selenium WebDriver
  - WebDriver Java
  - WebDriverManager
  - PageObject
- Selenium Grid
  - TestContainers

- **Introducción a Selenium**
- Selenium IDE
- Selenium WebDriver
  - WebDriver Java
  - WebDriverManager
  - PageObject
- Selenium Grid
  - TestContainers

- 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

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



- **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/>






- Selenium tiene tres componentes:

Proyecto		Descripción
Selenium IDE		Plugin Firefox 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

- Introducción a Selenium
- **Selenium IDE**
- Selenium WebDriver
  - WebDriver Java
  - WebDriverManager
  - PageObject
- Selenium Grid
  - TestContainers



- Selenium tiene tres componentes:

Proyecto		Descripción
Selenium IDE		Plugin Firefox 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

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



# 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

- Introducción a Selenium
- Selenium IDE
- **Selenium WebDriver**
  - WebDriver Java
  - WebDriverManager
  - PageObject
- Selenium Grid
  - TestContainers

- Selenium tiene tres componentes:

Proyecto		Descripción
Selenium IDE		Plugin Firefox 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

- 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/>

- WebDriver maneja de forma nativa los navegadores
- Necesita un **programa binario** que comunica la librería de **WebDriver** con el navegador (como un driver):
  - 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)

- Introducción a Selenium
- Selenium IDE
- **Selenium WebDriver**
  - **WebDriver Java**
  - WebDriverManager
  - PageObject
- Selenium Grid
  - TestContainers



- Importar la dependencia de **selenium-java** en nuestro proyecto (añadir al pom.xml)

```
<dependency>
  <groupId>org.seleniumhq.selenium</groupId>
  <artifactId>selenium-java</artifactId>
  <version>${selenium.version}</version>
  <scope>test</scope>
</dependency>
```

- **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)

- Crear el objeto WebDriver

```
WebDriver driver = new FirefoxDriver();  
WebDriver driver = new ChromeDriver();  
WebDriver driver = new OperaDriver();  
WebDriver driver = new InternetExplorerDriver();  
WebDriver driver = new EdgeDriver();  
WebDriver driver = new SafariDriver();
```

- Abrir una página web

```
driver.get("http://en.wikipedia.org/wiki/Main_Page");
```

- Localizar elementos en la página

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

// Locate element list
List<WebElement> webElements = driver.findElements(By...);
```

XPath es un lenguaje que permite seleccionar elementos dentro de documentos HTML y XML

```
/html/body/form[1]
//form[@id='loginForm']
//input[@name='username']
//form[@id='loginForm']/input[1]
```

<http://www.w3schools.com/xpath/>

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

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

String text = webElement1.getText();
String href = webElement1.getAttribute("href");
String css = webElement1.getCssValue("css");
Dimension dim = webElement1.getSize();

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

- En ocasiones necesitamos **esperar** a que ciertos elementos alcancen un resultado esperado
- Para ello utilizamos la clase **WebDriverWait** que nos permite definir cuál es la condición por la que debemos esperar

- Esperar a que ciertos elementos alcancen un estado determinado

```
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));  
  
wait.until(ExpectedConditions.elementToBeClickable(By.id("id1")));  
wait.until(ExpectedConditions.elementToBeSelected(By.id("id2")));  
wait.until(ExpectedConditions.presenceOfElementLocated(By.id("id3")));  
wait.until(ExpectedConditions  
    .textToBePresentInElementLocated(By.tagName("body"), "text"));  
wait.until(ExpectedConditions.titleIs("Page title"));
```

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

- **Test con Selenium y Google Chrome**
  - Se descarga el binario **chromedriver** de <https://sites.google.com/a/chromium.org/chromedriver/>
  - Se configura la propiedad del sistema **webdriver.chrome.driver** con la ruta donde está el binario
  - Se crea un objeto de la clase **ChromeDriver** para controlar el browser
  - Al finalizar el test se cierra el browser con el **método quit()**



# Selenium WebDriver: Java

```
public class ChromeTest {  
  
    protected WebDriver driver;  
  
    @BeforeAll  
    public static void setupClass() {  
        System.setProperty("webdriver.chrome.driver",  
            "/absolute/path/to/chromedriver");  
    }  
  
    @BeforeEach  
    public void setupTest() {  
        driver = new ChromeDriver();  
    }  
  
    @AfterEach  
    public void teardown() {  
        if (driver != null) {  
            driver.quit();  
        }  
    }  
  
    @Test  
    public void test() {  
        // Exercise and verify  
    }  
}
```

- **Testing Headless**

- Podemos ejecutar los test de Selenium sin lanzar una interfaz de usuario (navegador)
- Ideal cuando se ejecutan en máquinas sin interfaz visual (p.e. Sistemas de CI)

```
@BeforeEach
public void setupTest() {
    ChromeOptions options = new ChromeOptions();
    options.addArguments("--headless");
    driver = new ChromeDriver(options);
}
```

- Introducción a Selenium
- Selenium IDE
- **Selenium WebDriver**
  - WebDriver Java
  - **WebDriverManager**
  - PageObject
- Selenium Grid
  - TestContainers

- **Test con Selenium y Google Chrome**
  - La librería **webdrivermanager** se descarga automáticamente el driver y configurar la propiedad

```
<dependency>
  <groupId>io.github.bonigarcia</groupId>
  <artifactId>webdrivermanager</artifactId>
  <version>${webdrivermanager.version}</version>
  <scope>test</scope>
</dependency>
```

<https://github.com/bonigarcia/webdrivermanager>

# Selenium WebDriver: WebDriverManager

```
public class ChromeTest {  
  
    protected WebDriver driver;  
  
    @BeforeAll  
    public static void setupClass() {  
        WebDriverManager.chromedriver().setup();  
    }  
    @BeforeEach  
    public void setupTest() {  
        driver = new ChromeDriver();  
    }  
    @AfterEach  
    public void teardown() {  
        if (driver != null) {  
            driver.quit();  
        }  
    }  
    @Test  
    public void test () {  
        // Exercise and verify  
    }  
}
```

ejem1

# Selenium WebDriver: WebDriverManager

ejem1

```
@Test
public void test() throws InterruptedException {

    driver.get("https://wikipedia.org");
    WebElement searchInput = driver.findElement(By.name("search"));

    Thread.sleep(2000);

    searchInput.sendKeys("Rick Astley");
    searchInput.submit();

    Thread.sleep(2000);

    WebElement link = driver.findElement(By.linkText("Rickrolling"));
    link.click();

    Thread.sleep(2000);

    boolean memeFound = driver.findElements(By.cssSelector("p"))
        .stream()
        .anyMatch(element -> element.getText().contains("meme"));

    assertTrue(memeFound, "Rickrolling page should contain meme word");
}
```

- Selenium WebDriver con Firefox

```
public class FirefoxDriver {
    protected WebDriver driver;

    @BeforeAll
    public static void setupClass() {
        System.setProperty("webdriver.gecko.driver",
            "/absolute/path/to/geckodriver");
    }
    @BeforeEach
    public void setupTest() {
        driver = new FirefoxDriver();
    }
    @AfterEach
    public void teardown() {
        if (driver != null) {
            driver.quit();
        }
    }
    @Test
    public void test() {
        // Exercise and verify
    }
}
```

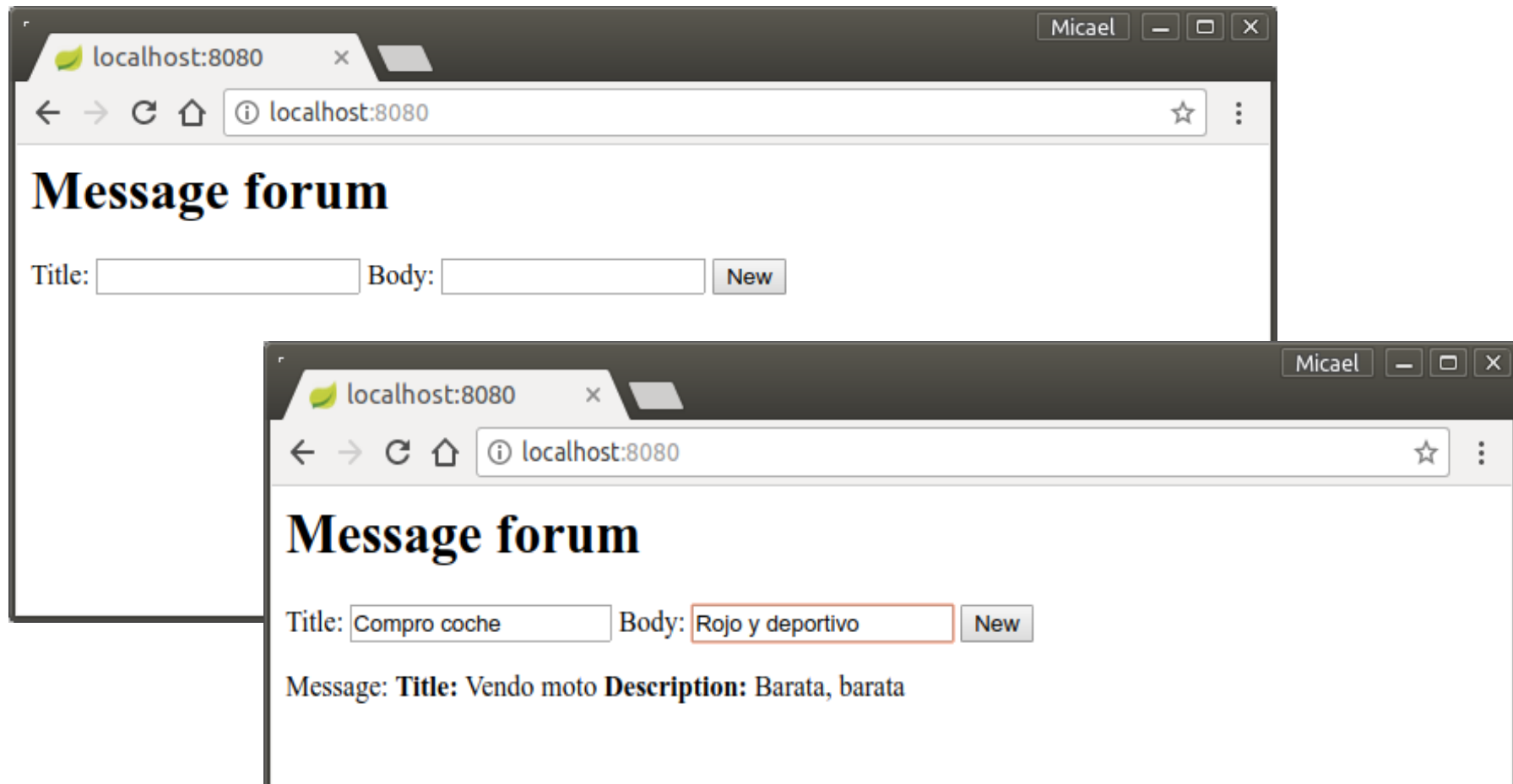
```
public class FirefoxDriver {
    protected WebDriver driver;

    @BeforeClass
    public static void setupClass() {
        WebDriverManager.firefoxdriver().setup();
    }

    @Before
    public void setupTest() {
        driver = new FirefoxDriver();
    }
    @AfterEach
    public void teardown() {
        if (driver != null) {
            driver.quit();
        }
    }
    @Test
    public void test () {
        // Exercise and verify
    }
}
```

ejem2

- Ejemplo





# Selenium WebDriver: WebDriverManager

ejem2

```
..<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 WebDriver: WebDriverManager

```
@SpringBootTest(classes = Application.class, webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
public class WebAppTest {
```

ejem2

```
    @LocalServerPort
    int port;
```

```
    private WebDriver driver;
```

```
    @BeforeAll
```

```
    public static void setupClass() {
        WebDriverManager.chromedriver().setup();
    }
```

```
    @BeforeEach
```

```
    public void setupTest() {
        driver = new ChromeDriver();
    }
```

```
    @AfterEach
```

```
    public void teardown() {
        if (driver != null) {
            driver.quit();
        }
    }
```

```
    @AfterEach
```

```
    public void teardown() {
        if (driver != null) {
            driver.quit();
        }
    }
```

```
    @Test
```

```
    public void test() { ... }
```

```
}
```

Configura la aplicación para que se lance en el contexto de este test

# Selenium WebDriver: WebDriverManager

ejem2

```
@Test
public void test() {

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

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

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

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

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

    assertThat(title).isEqualTo(newTitle);
    assertThat(body).isEqualTo(newBody);
}
```

- **Ejercicio 1: Tests de Web de Anuncios**
  - Implementa tests de selenium de la aplicación de Gestión de Anuncios
  - Crear y eliminar anuncios
  - Comprobar que el nombre del usuario aparece automáticamente en el segundo mensaje

- Podemos utilizar Selenium de manera análoga en Quarkus

ejem3

```
@Path("greeting")
public class GreetingResource {

    @Inject
    Template greetingTemplate;

    @GET
    @Produces(MediaType.TEXT_HTML)
    public TemplateInstance greeting(@QueryParam("name") String name) {
        return greetingTemplate.data("name", name);
    }
}
```

```
<html>

<body>
    <p id="greeting">Hello, {name}</p>
</body>

</html>
```

# Selenium Web Driver con Quarkus

ejem3

```
@QuarkusTest
public class GreetingTest {

    @ConfigProperty(name = "quarkus.http.test-port")
    int httpTestPort;

    private WebDriver driver;

    @BeforeAll
    public static void setupClass() { WebDriverManager.chromedriver().setup();}

    @BeforeEach
    public void setupTest() { driver = new ChromeDriver();}

    @AfterEach
    public void teardown() { if (driver != null) driver.quit();}

    @Test
    public void greetingTest() {

        String name = "Michel";

        driver.get("http://localhost:"+httpTestPort+"/greeting?name=" + name);

        String greeting = driver.findElement(By.id("greeting")).getText();

        assertThat("Hello, "+name).isEqualTo(greeting);

    }
}
```

# Selenium Web Driver con Quarkus

ejem3

@QuarkusTest

```
public class GreetingTest {  
  
    @ConfigProperty(name = "quarkus.http.test-port")  
    int httpTestPort;  
  
    private WebDriver driver;  
  
    @BeforeAll  
    public static void setupClass() { WebDriverManager.chromedriver().setup();}  
  
    @BeforeEach  
    public void setupTest() { driver = new ChromeDriver();}  
  
    @AfterEach  
    public void teardown() { if (driver != null) driver.quit();}  
  
    @Test  
    public void greetingTest() {  
        String name = "Michel";  
  
        driver.get("http://localhost:"+httpTestPort+"/greeting?name=" + name);  
  
        String greeting = driver.findElement(By.id("greeting")).getText();  
  
        assertThat("Hello, "+name).isEqualTo(greeting);  
    }  
}
```

Configura la aplicación para que se lance en el contexto de este test

# Selenium Web Driver con Quarkus

ejem3

```
@QuarkusTest
public class GreetingTest {

    @ConfigProperty(name = "quarkus.http.test-port")
    int httpTestPort;

    private WebDriver driver;

    @BeforeAll
    public static void setupClass() { WebDriverManager.chromedriver().setup();}

    @BeforeEach
    public void setupTest() { driver = new ChromeDriver();}

    @AfterEach
    public void teardown() { if (driver != null) driver.quit();}

    @Test
    public void greetingTest() {

        String name = "Michel";

        driver.get("http://localhost:"+httpTestPort+"/greeting?name=" + name);

        String greeting = driver.findElement(By.id("greeting")).getText();

        assertThat("Hello, "+name).isEqualTo(greeting);

    }
}
```

Obtenemos el puerto de la aplicación lanzada en el contexto del test (evitamos utilizar el puerto del modo desarrollo)



# Selenium Web Driver con Quarkus

ejem3

```
@QuarkusTest
public class GreetingTest {

    @ConfigProperty(name = "quarkus.http.test-port")
    int httpTestPort;

    private WebDriver driver;

    @BeforeAll
    public static void setupClass() { WebDriverManager.chromedriver().setup();}

    @BeforeEach
    public void setupTest() { driver = new ChromeDriver();}

    @AfterEach
    public void teardown() { if (driver != null) driver.quit();}

    @Test
    public void greetingTest() {
        String name = "Michel";

        driver.get("http://localhost:"+httpTestPort+"/greeting?name="

        String greeting = driver.findElement(By.id("greeting")).getText();

        assertThat("Hello, "+name).isEqualTo(greeting);
    }
}
```

Configuración análoga al  
ejemplo de con  
SpringFramework

# Selenium Web Driver con Quarkus

```
@QuarkusTest
public class GreetingTest {

    @ConfigProperty(name = "quarkus.http.test-port")
    int httpTestPort;

    private WebDriver driver;

    @BeforeAll
    public static void setupClass() { WebDriverManager.chromedriver().setup(); }

    @BeforeEach
    public void setupTest() { driver = new ChromeDriver(); }

    @AfterEach
    public void teardown() { if (driver != null) driver.quit(); }


    @Test
    public void greetingTest() {
        String name = "Michel";

        driver.get("http://localhost:"+httpTestPort+"/greeting?name=" + name);

        String greeting = driver.findElement(By.id("greeting")).getText();

        assertThat("Hello, " + name).isEqualTo(greeting);
    }
}
```

Testing de la funcionalidad

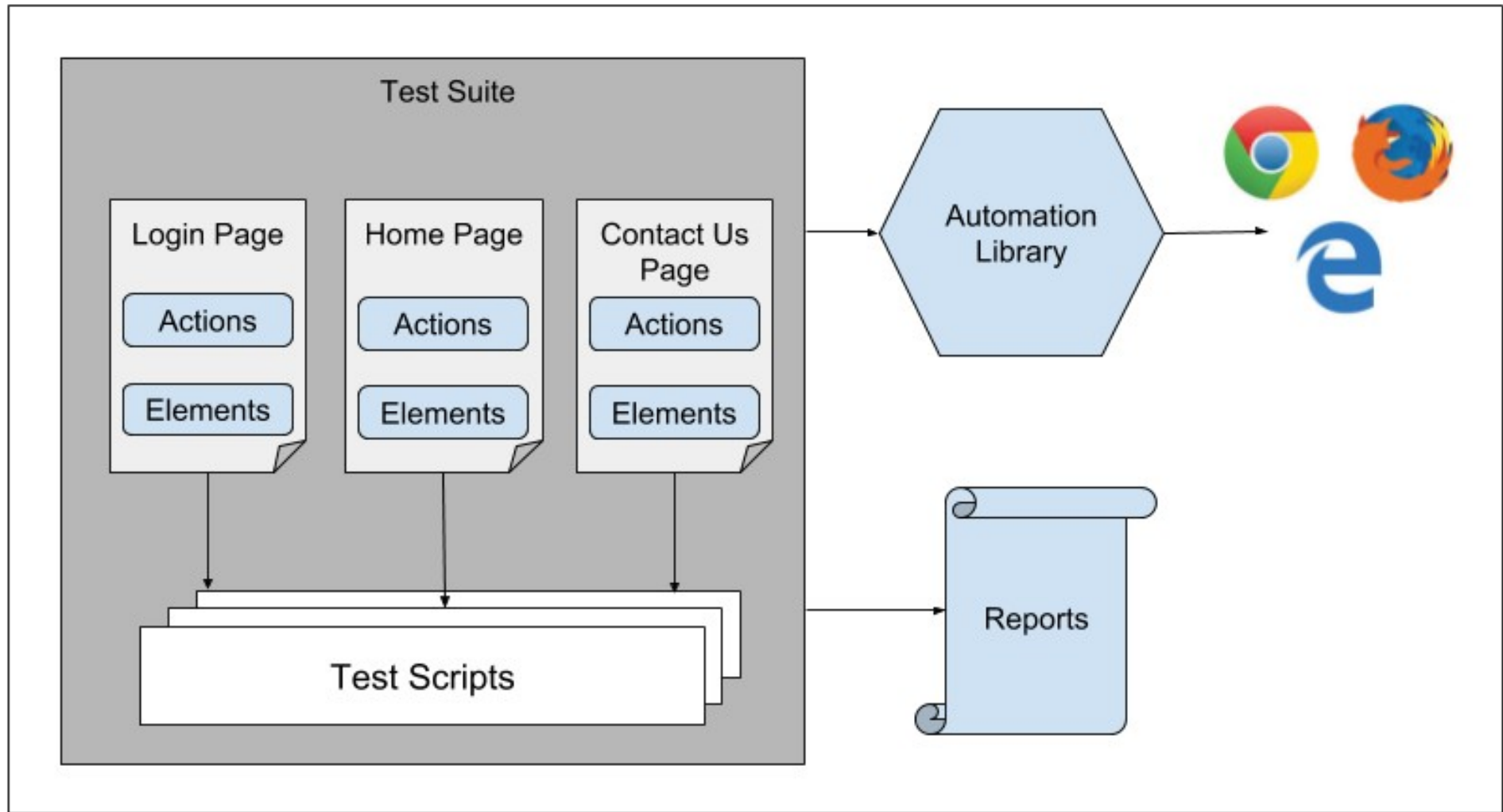


- Introducción a Selenium
- Selenium IDE
- **Selenium WebDriver**
  - WebDriver Java
  - WebDriverManager
  - **PageObject**
- Selenium Grid
  - TestContainers

- **La interfaz** es la parte visible de nuestra aplicación y la que más feedback recibe, por lo que **es la más susceptible a los cambios**.
- Por lo tanto, **los test de interfaz no suelen ser estables** y cambian constantemente.
- Es difícil recordar todos los test que utilizan un componente que acabamos de modificar (cambian etiquetas, clases y Ids del HTML)

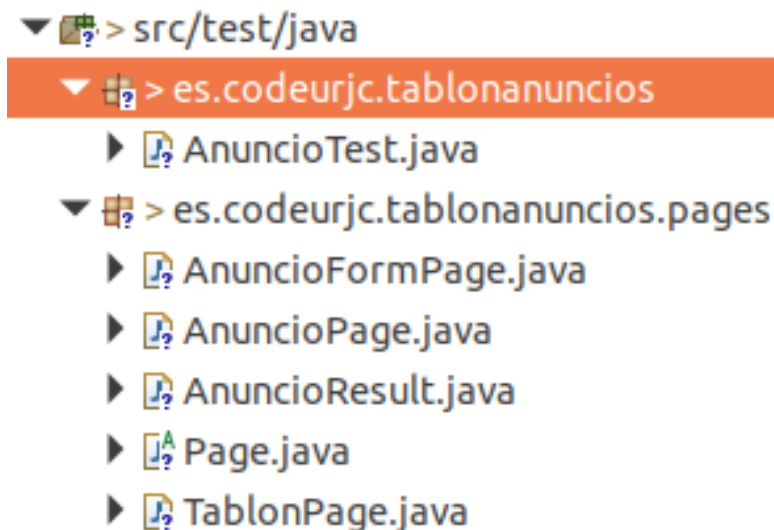
- Para abordar problemas como este, surge el patrón de diseño **PageObject**.
- Consiste en crear una clase que represente un conjunto de elementos de la interfaz.
- Normalmente se utiliza para **agrupar toda la lógica de interacción de una página**.
- Pero también puede utilizarse para agrupar componentes visuales reutilizables, como los menus.

# Selenium Web Driver: PageObject



# Selenium Web Driver: PageObject

- Veamos un ejemplo de PageObject sobre nuestra aplicación de anuncios.
- Agruparemos los elementos web en las páginas que devuelven los controladores.



# Selenium Web Driver: PageObject

ejem4

```
@Test
public void createAnuncio() throws InterruptedException {
    driver.get("http://localhost:8080/");

    driver.findElement(By.linkText("Nuevo anuncio")).click();

    driver.findElement(By.name("nombre")).sendKeys("Anuncio nuevo con Selenium");
    driver.findElement(By.name("asunto")).sendKeys("Vendo moto");
    driver.findElement(By.name("comentario")).sendKeys("Un comentario muy largo...");

    driver.findElement(By.xpath("//input[@type='submit']")).click();

    driver.findElement(By.linkText("Volver al tablón")).click();

    assertNotNull(driver.findElement(By.partialLinkText("Selenium")));
}
```

Con Selenium

```
@Test
public void createAnuncio() throws InterruptedException {
    TablonPage tablon = new TablonPage(driver, port);

    String name = "Michel";
    String asunto = "Vendo coche";
    String comentario = "Vendo Opel Corsa";

    tablon.get() // -> Go to TablonPage
        .nuevoAnuncio() // -> Go to AnuncioFormPage
            .rellenarNuevoAnuncio(name, asunto, comentario)
            .enviarNuevoAnuncio() // -> Go to AnuncioResult
        .volverAlTablon() // -> Go to TablonPage
            .comprobarNuevoAnuncio(name, asunto);
}
```

Con Selenium + PageObject



# Selenium Web Driver: PageObject

- A través de la herencia, podemos delegar mucha lógica en una **clase padre** (que denominaremos *Page*) que cuente con métodos auxiliares reutilizables.

```
public abstract class Page {  
  
    protected WebDriver driver;  
    protected WebDriverWait wait;  
    protected int port;  
  
    public Page(WebDriver driver, int port) {  
        this.driver = driver;  
        this.port = port;  
        this.wait = new WebDriverWait(driver, Duration.ofSeconds(10));  
    }  
  
    protected void get(String path) {  
        driver.get("localhost:"+this.port+path);  
    }  
  
    protected boolean isElementPresent(String text) { ... }  
  
    protected WebElement findElementWithText(String text) { ... }  
  
    protected List<WebElement> findElementsWithText(String text) { ... }  
  
    protected By getConditionForText(String text) { ... }  
  
}
```

ejem4

# Selenium Web Driver: PageObject

- Quedando clases que **encapsulan las acciones** sobre elementos de una página.

ejem4

```
public class AnuncioFormPage extends Page{

    public AnuncioFormPage(WebDriver driver, int port) {
        super(driver, port);
    }

    public AnuncioFormPage get(){
        this.get("/nuevo_anuncio");
        wait.until(ExpectedConditions.elementToBeClickable(By.tagName("h1")));
        return this;
    }

    public AnuncioFormPage rellenarNuevoAnuncio(String nombre, String asunto, String comentario) {
        driver.findElement(By.name("nombre")).sendKeys(nombre);
        driver.findElement(By.name("asunto")).sendKeys(asunto);
        driver.findElement(By.name("comentario")).sendKeys(comentario);
        return this;
    }

    public AnuncioResult enviarNuevoAnuncio() {
        driver.findElement(By.xpath("//input[@type='submit']")).click();
        return new AnuncioResult(this);
    }

}
```

- Además, podemos reutilizar estos métodos y clases para elaborar casos de prueba distintos.

ejem4


```
@Test
public void deleteAnuncio() throws InterruptedException {
    TablonPage tablon = new TablonPage(driver, port);

    String nombre = "Pepe";
    String asunto = "Hola caracola";

    tablon.get() // -> Go to TablonPage
        .verAnuncio(nombre, asunto) // -> Go to AnuncioPage
        .borrarAnuncio() // -> Go to AnuncioResult
        .volverAlTablon() // -> Go to TablonPage
        .comprobarAnuncioBorrado(nombre, asunto);
}
```

- Introducción a Selenium
- Selenium IDE
- Selenium WebDriver
  - WebDriver Java
  - WebDriverManager
  - PageObject
- **Selenium Grid**
  - TestContainers

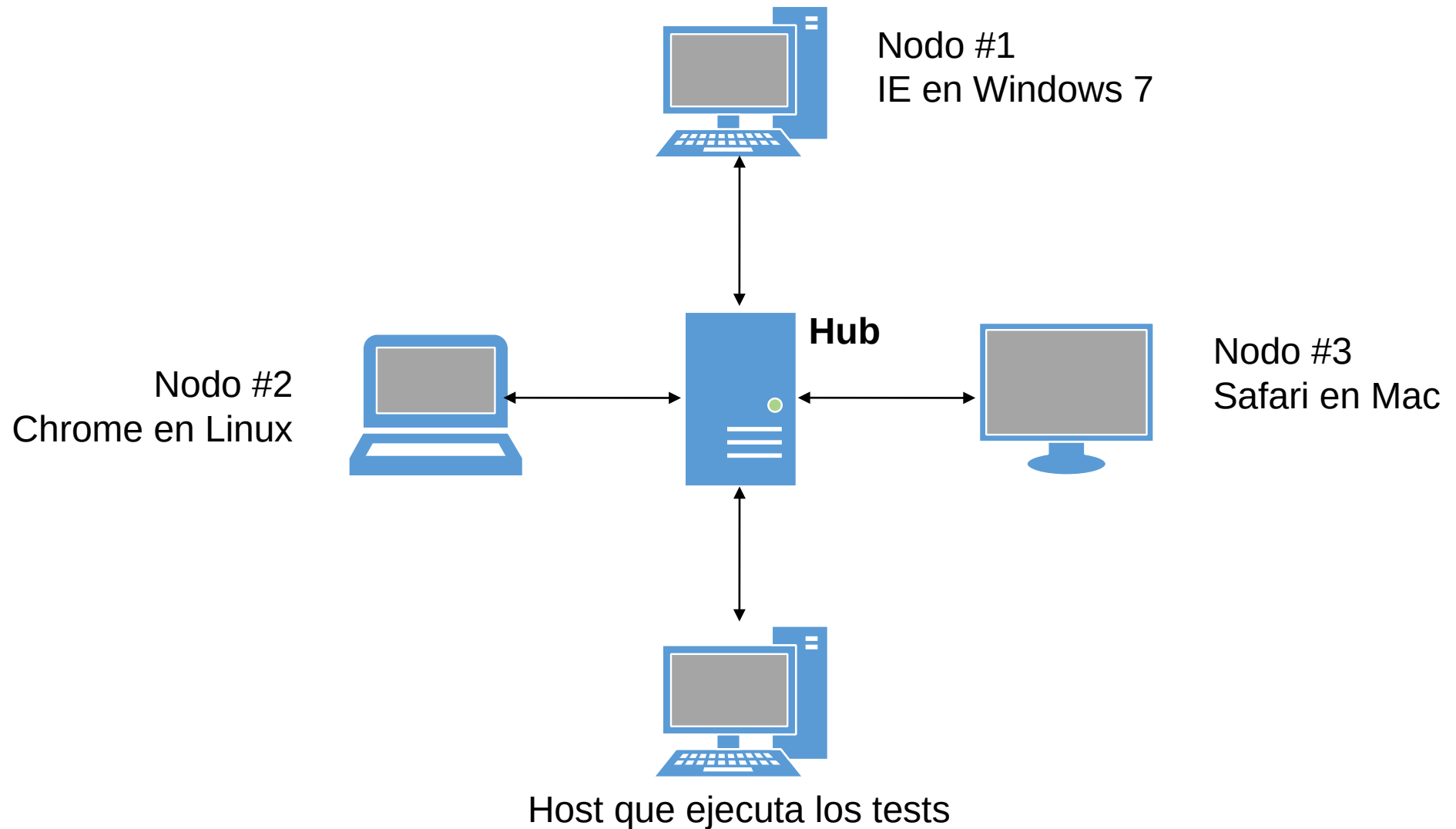
- Selenium tiene tres componentes:

Proyecto		Descripción
Selenium IDE		Plugin Firefox 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

- 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



- **Hub**

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

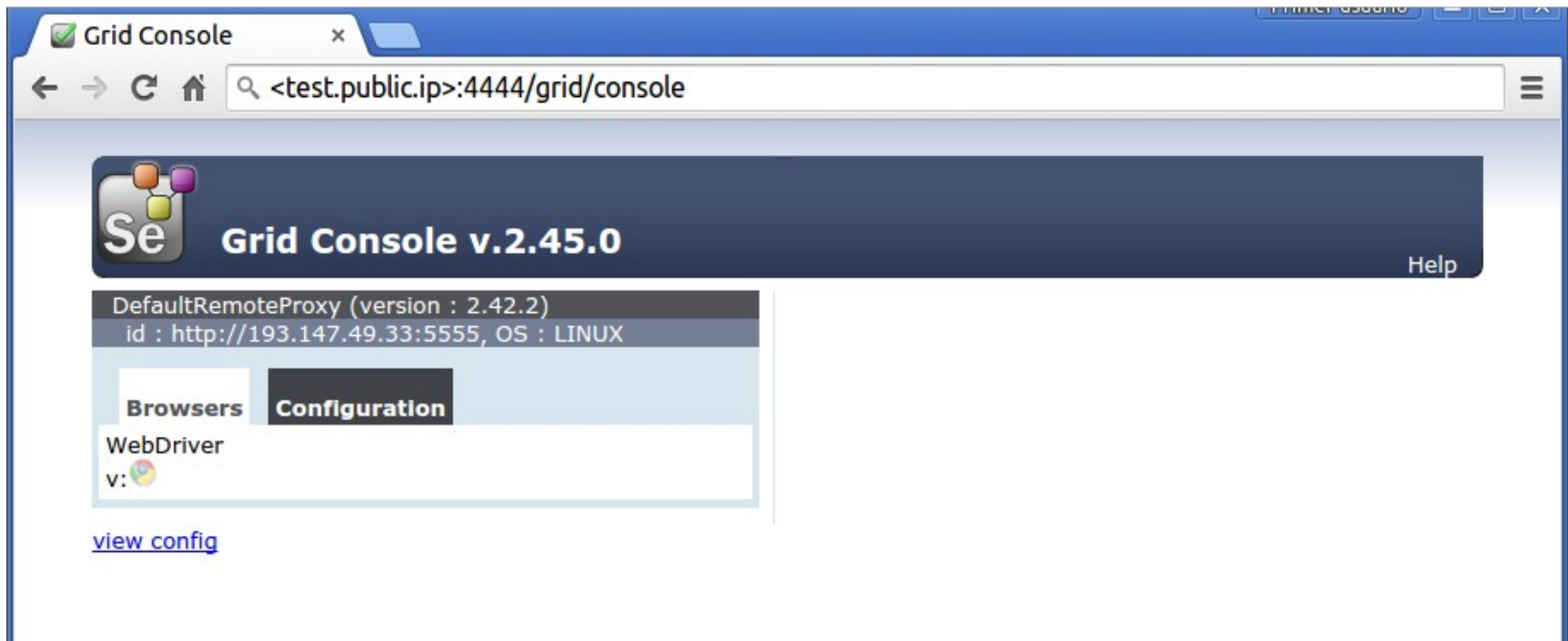
- **Nodos**

```
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/>



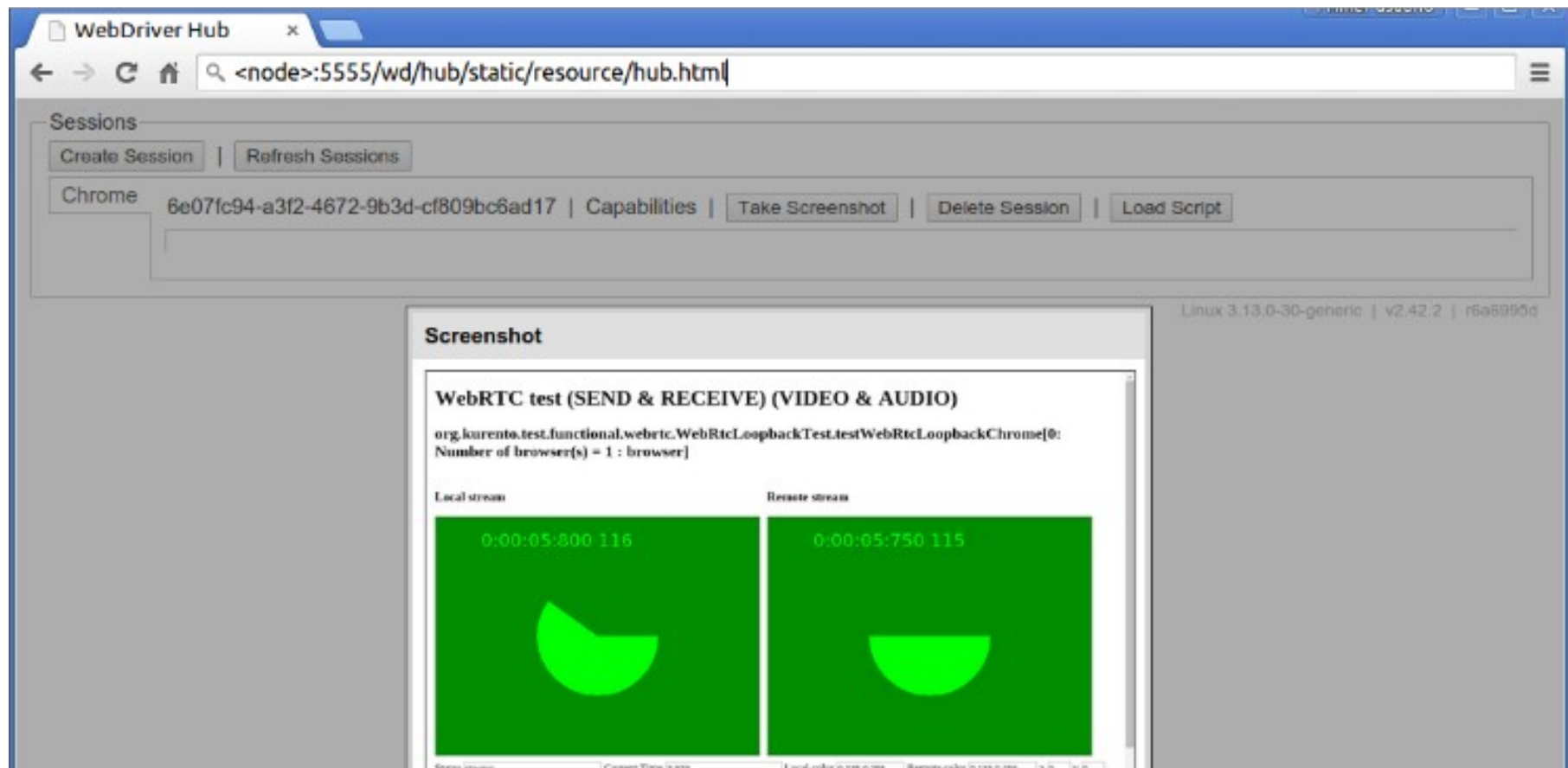
- Consola de administración del Hub



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

# Selenium Grid

- Consola de administración del nodo



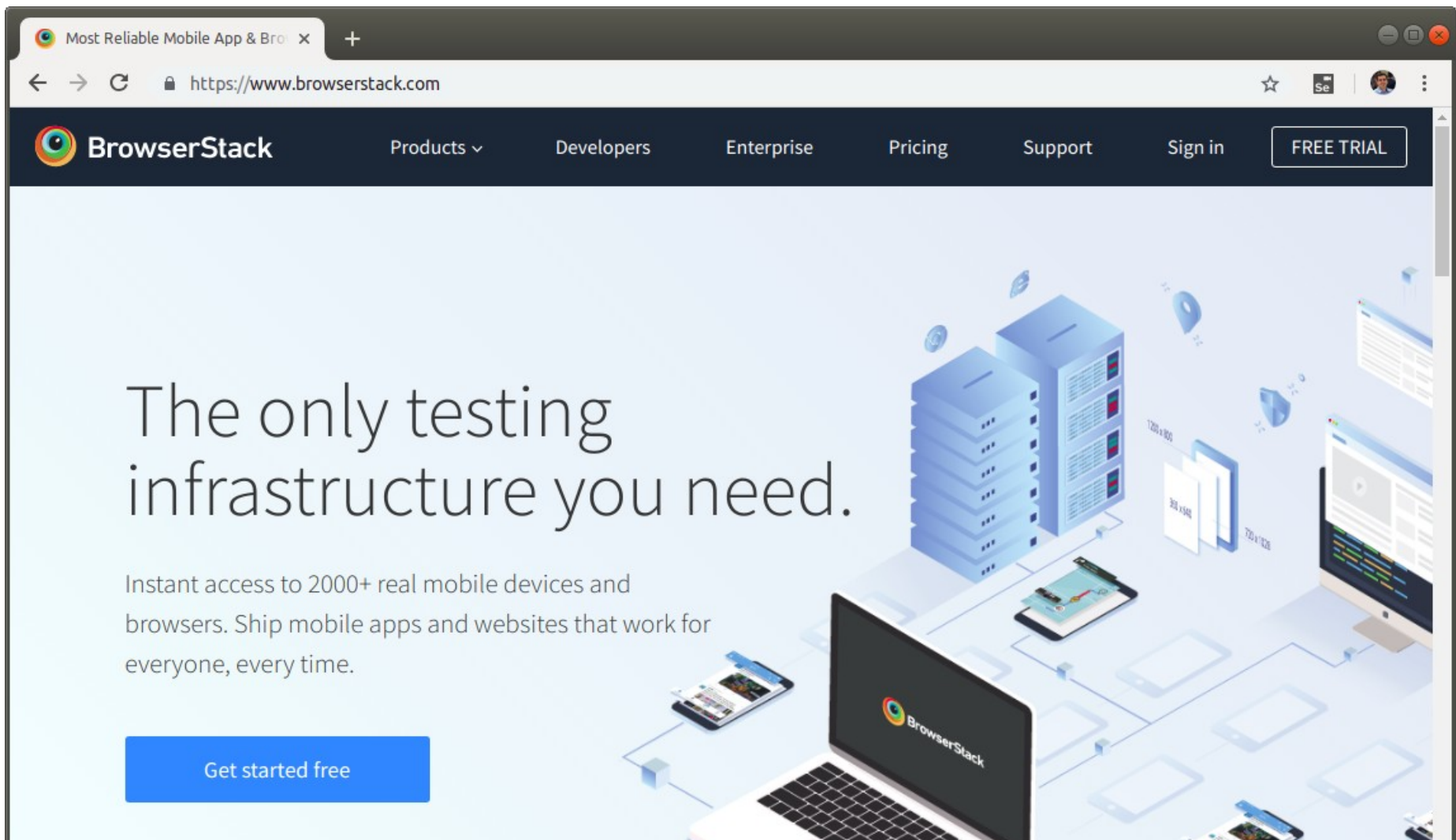
<http://<node-address>:<node-port>/wd/hub/static/resource/hub.html>

- La conexión desde el test usa la clase **RemoteWebDriver**

```
public class RemoteChromeTest {  
  
    protected WebDriver driver;  
  
    @BeforeEach  
    public void setup() {  
        DesiredCapabilities caps = new DesiredCapabilities().chrome();  
        driver = new RemoteWebDriver(new URL("http://hub-server:hub-port/"), caps);  
    }  
  
    @AfterEach  
    public void teardown() {  
        if (driver != null) {  
            driver.quit();  
        }  
    }  
  
    @Test  
    public void test() {  
        // Exercise and verify  
    }  
}
```

# Selenium Grid

- Servicios online que ofrecen navegadores web



The screenshot shows the BrowserStack website in a web browser. The browser's address bar displays 'https://www.browserstack.com'. The website's navigation bar includes the BrowserStack logo, links for 'Products', 'Developers', 'Enterprise', 'Pricing', 'Support', 'Sign in', and a 'FREE TRIAL' button. The main content area features the headline 'The only testing infrastructure you need.' followed by the text 'Instant access to 2000+ real mobile devices and browsers. Ship mobile apps and websites that work for everyone, every time.' and a blue 'Get started free' button. The background of the main content area is an isometric illustration showing a laptop with the BrowserStack logo, several mobile phones, and server racks, all interconnected by lines representing a network or testing infrastructure.

- Introducción a Selenium
- Selenium IDE
- Selenium WebDriver
  - WebDriver Java
  - WebDriverManager
  - PageObject
- **Selenium Grid**
  - **TestContainers**

- Es posible **simular un navegador remoto** por medio de la librería TestContainers.
- Recordemos que para usar esta librería, necesitamos **tener Docker instalado**.
- Muy útil si no tenemos todos los navegadores instalados o necesitamos probar **distintas versiones**.

- Debemos añadir las siguientes dependencias:

```
...  
<dependency>  
  <groupId>org.seleniumhq.selenium</groupId>  
  <artifactId>selenium-chrome-driver</artifactId>  
  <version>4.7.2</version>  
</dependency>  
<!-- TEST CONTAINER DEPENDENCIES -->  
<dependency>  
  <groupId>org.testcontainers</groupId>  
  <artifactId>testcontainers</artifactId>  
  <version>${org.testcontainer.version}</version>  
</dependency>  
<dependency>  
  <groupId>org.testcontainers</groupId>  
  <artifactId>junit-jupiter</artifactId>  
  <version>${org.testcontainer.version}</version>  
  <scope>test</scope>  
</dependency>  
<dependency>  
  <groupId>org.testcontainers</groupId>  
  <artifactId>selenium</artifactId>  
  <version>${org.testcontainer.version}</version>  
  <scope>test</scope>  
</dependency>
```

ejem5

# Selenium Grid: TestContainers

```
@Testcontainers
public class WikipediaTest {

    @Container
    public static BrowserWebDriverContainer<?> seleniumContainer = new BrowserWebDriverContainer<>()
        .withRecordingMode(RECORD_ALL, new File("target"), VncRecordingFormat.MP4);

    private RemoteWebDriver driver;

    @BeforeEach
    public void setupTest() {
        driver = new RemoteWebDriver(seleniumContainer.getSeleniumAddress(), new ChromeOptions());
    }

    @AfterEach
    public void teardown() {
        if (driver != null) { driver.quit();}
    }

    @Test
    public void test() throws InterruptedException {...}

}
```

ejem5



- **Ejercicio 4: Tests de Web de Mensajes con Test Containers**
  - Modifica el ejemplo 2 (*selenium\_ejem2*) para que utilice un navegador proporcionado por TestContainers
  - **NOTA:** El navegador de TestContainers no podrá acceder al localhost de nuestra máquina

<https://www.testcontainers.org/features/networking/#exposing-host-ports-to-the-container>