



# MDAS

Máster en Desarrollo y Arquitectura Software

FUNDAMENTOS DE PRUEBAS - 2024

# Prerrequisitos

- OpenJDK = 21 desde <https://adoptopenjdk.net/>
- IDE: IntelliJ community edition <https://www.jetbrains.com/idea/download>
- Browser: Firefox/Chrome

## Objetivos – Día I

- Entender qué es Selenium y qué nos permite hacer.

# Introducción a Selenium Tool - Componentes

¿Qué es? <https://www.selenium.dev/>

 Selenium WebDriver

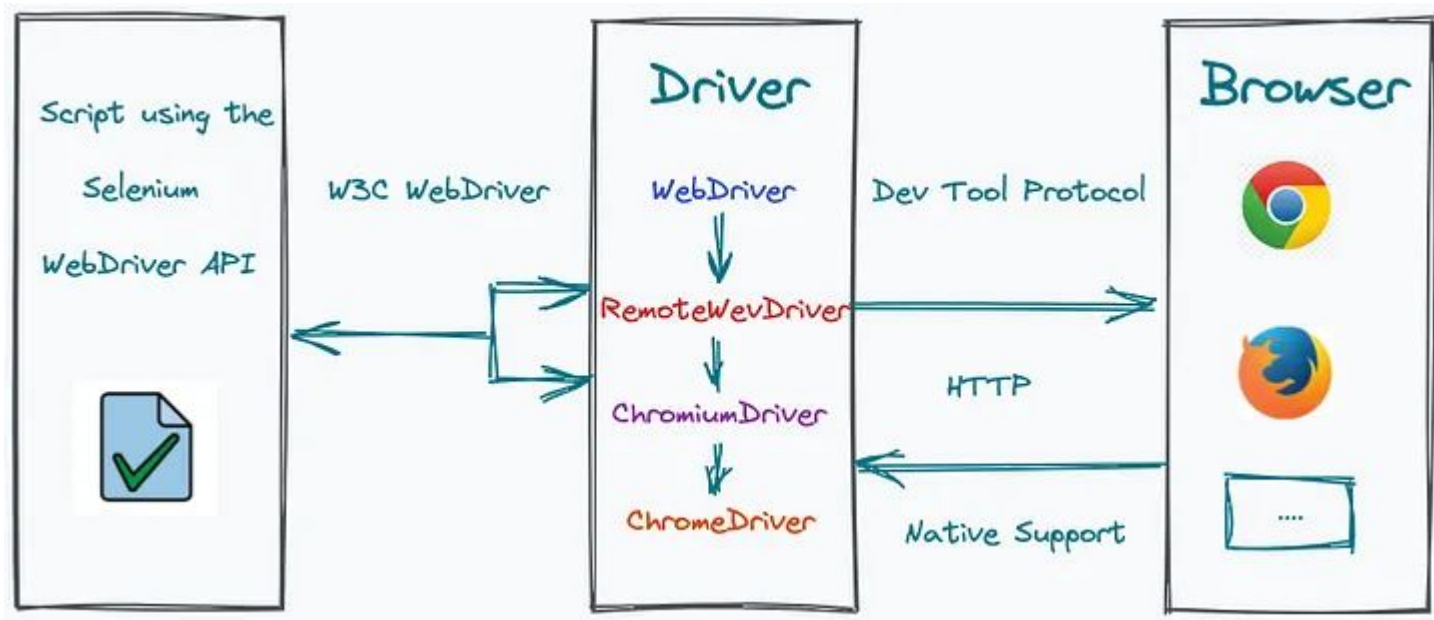
 Selenium IDE

 Selenium Grid



# Introducción a Selenium Tool - Architecture

- [World Wide Web Consortium \(W3C\) standard.](#)
- [DevTools Protocol](#)



# Introducción a Selenium Tool – Selenium IDE: práctica

- Instalar Selenium IDE en Chrome/Firefox
- Sobre <https://www.selenium.dev/> automatizar navegación a *Documentación > Selenium IDE*
- Sobre <https://the-internet.herokuapp.com> automatizar “Drag & Drop”

# Introducción a Selenium Tool – Funcionamiento

- WebDriver:
  - Window: get, getTitle, getCurrentUrl, getPageSource (Tip: <https://jsoup.org/>), close, quit
  - Navigate: to, back, forward, refresh
  - FindElement & FindElements
  - switchTo: frame, alert, window...
- Locators, findElementBy:
  - By Id
  - By name
  - By css: <https://saucelabs.com/resources/articles/selenium-tips-css-selectors>
  - By xpath
- WebElement: click, clear, findElement/s, getAttribute, getText, sendkeys...

# Introducción a Selenium Tool – Wait a minute

**Selenium** siempre intentará encontrar los elementos definidos en una página después del evento “pageLoad”. En caso de no encontrarlo, tenemos dos opciones:

- Espera implícita
- Espera explícita

Documentación: <https://www.selenium.dev/documentation/webdriver/ waits/>



# Introducción a Selenium Tool – Wait a minute

- Las esperas implícitas se utilizan para proporcionar un tiempo de espera en todas las peticiones para hacer un segundo intento para encontrar el elemento:
  - `driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);`
  - En caso de no cumplirse saltará la excepción: "No Such Element Exception".
  - **Por defecto este tiempo es 0.**
- Las esperas explícitas se utilizan para detener la ejecución hasta que se cumpla una condición particular o haya transcurrido el tiempo máximo.
  - Se puede combinar `WebDriverWait`, `ExpectedConditions` y `FluentWait` (clase `Wait`):
    - `driver.until(ExpectedConditions.visibilityOfElementLocated(By...));`
  - En caso de no cumplirse saltará la excepción: 'TimeoutException'
  - Por defecto intentan encontrar el elemento cada 0,5 segundos

# Selenium WebDriver: práctica – preparación

Ver [s2oBCN/laSalle \(github.com\)](https://github.com/s2oBCN/laSalle)

- Test sobre <https://the-internet.herokuapp.com>

## Selenium WebDriver: práctica – Levantar el navegador

- `System.setProperty ("webdriver.chrome.driver","full path to driver exe" );`
- `driver = new ChromeDriver();`
- `driver.manage().window().maximize();`
- `LOGGER.debug("driver started");`
  
- `driver.close();`
- `LOGGER.debug("driver closed");`

## Selenium WebDriver: práctica – Navegación

- `driver.get("https://the-internet.herokuapp.com");`
- `getTitle`
- `getPageSource`
- `navigate()`

## Selenium WebDriver: práctica – Encontrar elementos

- `driver.navigate().to("https://the-internet.herokuapp.com");`
- `findElement:`
  - `By.id`
  - `linkText: "JavaScript Alerts"`
  - `cssSelector`
  - `xpath`

# Selenium WebDriver: práctica – Esperas FluentWait

- `driver.get("https://the-internet.herokuapp.com/dynamic_controls");`
- `driver.findElement(By.cssSelector("#checkbox-example > button")).click();`
- `Wait<WebDriver> fluentWait = new FluentWait<WebDriver>(driver)  
 .withTimeout(Duration.of(60, ChronoUnit.SECONDS))  
 .pollingEvery(Duration.of(2, ChronoUnit.SECONDS))  
 .ignoring(Exception.class);`
- `WebElement fluentElement = fluentWait.until(new Function<WebDriver, WebElement>() {  
 @Override  
 public WebElement apply(WebDriver webDriver) {  
 return webDriver.findElement(By.id("message"));  
 }  
});`

## Selenium WebDriver: práctica – Esperas “implicitWait”

- `driver.get("https://the-internet.herokuapp.com/dynamic_controls");`
- `driver.manage().timeouts().implicitlyWait(60, TimeUnit.SECONDS);`
- `driver.findElement(By.cssSelector("#checkbox-example > button")).click();`

## Selenium WebDriver: práctica – Esperas “explicitWait”

- `driver.get("https://the-internet.herokuapp.com/dynamic_loading/1");`
- `driver.findElement(By.cssSelector("#start > button")).click();`
- `WebDriverWait wait = new WebDriverWait(driver, 10);`
- `WebElement element`  
`=wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("finish")));`



## Objetivos – Día 2

- Poder crear un proyecto Java siguiendo:
  - BUENAS PRÁCTICAS

## Construyendo un framework

- Construir el CI antes que el framework de testing



# Construyendo un framework

- Mantenibilidad: se empieza con un test... ¿cuantos tendremos dentro de un año?
- Configuración: ejecución en diferentes entornos.
- Generación de informes de las pruebas realizadas.
- Paralelización: reducción tiempos de ejecución.
- Disponibilidad: los desarrolladores lo necesitara en cualquier momento.

# Construyendo un framework de UI testing

- Clean code:  
<https://medium.com/mindorks/how-to-write-clean-code-lessons-learnt-from-the-clean-code-robert-c-martin-9ffc7aef870c>
- [s2oBCN/test-academy-selenium \(github.com\)](https://github.com/s2oBCN/test-academy-selenium)
- Estructura siguiendo agrupación funcional,

# Construyendo un framework – Patrones

- Page Object Model
- Facade
- Builder

## Construyendo un framework – Referencias

- Todo en uno: [Serenity BDD](#), quickstarted [serenity-bdd/flying-high-tutorial \(github.com\)](#)
- WebDriver: [WebDriverManager](#)
- Reports: [Allure](#) (<https://github.com/allure-examples>, [allure-annotations](#))
- Configuración: <https://github.com/lightbend/config>
- Waits: <https://github.com/FluentLenium/FluentLenium>
- [Spring Framework](#)
- [Selenium tutorial](#)

# Construyendo un framework – Práctica

Utilizando los patrones aprendidos y frameworks Open Source. Implementar el siguiente escenario:

En la web de Vueling, verificar que existen vuelos para la siguiente búsqueda:

- Origen 'Madrid'
- Destino 'Barcelona',
- Fecha '1 Junio'
- Solo ida
- Un único pasajero

- Se puede implementar en cualquier lenguaje.
- El entregable debe contener:
  - Código fuente del proyecto (sin el compilado)
  - Un README con las instrucciones necesarias para la ejecución del test.
  - Un **report** con el resultado de la ejecución.