Sistemas Distribuidos e Internet

Tema 4

Introducción al Web Testing (Selenium)

Qué es el Web testing

- Es la prueba de una aplicación web para la detección de posible fallos antes de que sea desplegada en su entorno de producción.
- Tests posibles:
  - Test funcional
  - Test de usabilidad
  - Test de interface
  - Test de compatibilidad
  - Test de rendimiento
  - Test de seguridad

Tipos de tests – I

- Funcional: Prueba de todos los enlaces web, conexiones a bases de datos, envío y recepción de datos de formularios, cookies, …
- Usabilidad: Es el proceso mediante el que se miden las características de la interacción computador-humano, por lo que las debilidades de esta interacción deben identificarse para corregirse.
- Interface: Se refiere a las conexiones entre el servidor de aplicaciones y el servidor web, y el servidor de aplicaciones y el Servidor de base de datos.
- Compatibilidad: Navegadores, SSOO, Disp. Móviles e Impresión.

Tipos de test - II

- Rendimiento:
  - Pruebas de carga: un volumen de usuarios/conexiones, datos gestionados, conexiones a la BD, alta carga en páginas concretas
  - Pruebas de estrés: Exponer al sistema a valores límite de demanda de recursos y ver como responde. Los puntos críticos suelen ser campos de entrada, y areas de registro y login.
- Seguridad:
  - Uso de URLs internos directos sin identificarse.
  - Acceder a URLs para otro rol diferente al que se está identificado.
  - Ver la reacción a valores incorrectos en los formularios de login.
  - Acceso a directorios de recursos de descarga sin acceder a los enlaces de descarga.
  - Test CAPTCHA para scripts de login automático.
  - Test SSL, aviso cuando se accede a un URL https desde URLs http (no segura) y viceversa.
  - Todas las transacciones, mensaje de error y avisos de seguridad deben quedar reflejados en los archivos de log.

Qué vamos a hacer nosotros

- Elegir una herramienta que nos permita de una forma productiva realizar todos estos tipos de tests.
- ¿Qué tests vamos a diseñar?: Pues nos centraremos en los funcionales, pero también veremos de seguridad y de interface.

Elección de herramientas OpenSource

- Framework de pruebas unitarias
  - **JUnit**, TestNG, Mocking, Matching, Spock.
- Framework de pruebas funcionales
  - **Selenium**2/**3**, CasperJS, Capedit, Kantoo, Phantommjs, TestCafe, …
- Navegador
  - **Firefox**, Edge, Chrome, Firefox

Pruebas Unitarias: Junit vs TestNG

Similitudes

- OpenSource
- TestNG fue inspirado en Junit
- Son las librerías de testing más populares (ambas en el top 20)
- Junit presente en el 62% de los proyectos Java y TestNG en el 6%.

Diferencias

- JUnit: Más es más maduro y presenta una comunidad mucho mayor.
- testNG: Más potente

```
@Test(threadPoolSize = 3, invocationCount = 9)
public void testSomething() {
  ...
}
```

Pruebas funcionales

Elección de la versión de Selenium:

- Selenium2: Su configuración es muy simple ya que incorpora en la propia librería los drivers para los navegadores más populares (WebDriver).
- Selenium3: exige instalar un driver especifico según la subversión x de Selenium3.x y la versión de navegador, además de un pequeña configuración.
  - Por ejemplo para Selenium 3.141/Firefox94 y superiores se debe instalar geckodriver 0.30

→ Se ha seleccionado **Selenium 3.141.59**

Elección de navegador:

- Ambas versiones de Selenium disponen de soporte para los navegadores más populares: **Firefox**, Chrome, Edge y Safari.
- Elegiremos Firefox por ser la alternativa de Referencia de Selenium 2/3.

→ Se ha seleccionado **Firefox 94 o superior**

Componentes de Selenium

- **Selenium IDE**. Extensión de Firefox que permite grabar, editar y depurar pruebas. Permite exportar las pruebas grabadas a código **Selenase** (nativo de Selenium) o bien código Java, Ruby, Python y C# basado en Selenium API Client.
- ***Selenium API Client***. API para interactuar con Selenium desde código cliente.
  - **Selenium 3 presenta una nueva API basada en WebDriver**.
- ***Selenium WebDriver***. Componente de Selenium 3 API Cliente. Controlador del navegador que permite enviar comandos al propio navegador para realice acciones como si de un usuario real se tratase.
- **Selenium Grid.** Servidor que permite usar instancias de navegador ejecutándose en máquinas remotas.

API Selenium

https://www.seleniumhq.org/docs/03_webdriver.jsp

- WebDriver: Es la clase de la API de Selenium que encapsula la interacción con el navegador.

```
WebDriver driver  = FirefoxDriver();
driver.get("http://www.google.com");
```

- Dispone de drivers para múltiples navegadores:

  Emulación de un navegador HTML

  – WebDriver driver = **new** HtmlUnitDriver();

  Conexión real al navegador Firefox

  – WebDriver driver = **new** FirefoxDriver();

API Selenium - II

- Localización de un elemento de interfaz mediante WebDriver usando alguno de los métodos de búsqueda de WebDriver: findElement or findElements:
  - El método puede retornar a un WebElement o List<WebElement>.
  - En caso de no encontrar generar una excepción.

API Selenium – III (Localizadores)

- By.id
  - ...
  - WebElement element = driver.findElement(By.id("coolestWidgetEvah"));
- By.className
  - Cheddar Gouda
  - List<WebElement> cheeses = driver.findElements(By.className("cheese"));
- By.tagName, By.name, By.LinkText, By.partialLinkText, By.cssSelector, By.xpath

API Selenium - IV

- Como Obtener el valor de un campo input

```
WebElement dni = driver.findElement(By.id("dni"));
dni.getText();
```

- Como rellenar un campo input

```
WebElement dni = driver.findElement(By.id("dni"));
dni.click();
dni.clear();
dni.sendKeys(dnip);
```

- Como clickar un botón Submit o un enlace

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

API Selenium – V (Navegación)

- El método navigate:
  - Ir a una página:
    driver.navigate().to("http://www.example.com");
  - Ir adelante y atrás:

    driver.navigate().forward();

    driver.navigate().back();
- Cookies:
  - Create a cookie

```
Cookie cookie = new Cookie("key", "value");
driver.manage().addCookie(cookie);
```

  - Manage cookies

```
// By Cookie
driver.manage().deleteCookieNamed("CookieName");
// By Cookie
driver.manage().deleteCookie(loadedCookie);
// Or all of them
driver.manage().deleteAllCookies();
```

Esperas explícitas e implícitas

- Espera explicita:
  - Consiste en una espera por una condición antes de continuar la ejecución.
  - **El peor de los casos es la espera incondicional Thread.sleep() → PROHIBIDA EN ESTA ASIGNATURA.**
  - Con Selenium se combinan WebDriverWait con ExpectedCondition

```
WebDriver driver = new FirefoxDriver();
driver.get("http://somedomain/url_that_delays_loading");
WebElement myDynamicElement = (new WebDriverWait(driver, 10))
.until(ExpectedConditions.presenceOfElementLocated(By.id("my
DynamicElement")));
```

Esperas explícitas e implícitas

- Espera implícita:
  - Le dice a WebDriver que sondee el árbol DOM cada cierto tiempo para encontrar un elemento sino está disponible.

```
WebDriver driver = new FirefoxDriver();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
driver.get("http://somedomain/url_that_delays_loading");
WebElement myDynamicElement = driver.findElement(By.id("myDynamicElement"));
```

Selenium para Spring Boot

- Dos opciones:
  - Dos proyectos separados:
    - El proyecto a probar: un proyecto Spring Boot
    - El proyecto con las pruebas: proyecto Java con Selenium y Junit
      - Se incluyen las dependencias Maven para
        - » Junit 5.
        - » selenium-server3.141.59.jar
  - Un proyecto único: (Se ejecuta como dos proyecto diferentes).
    - Con el código Spring Boot donde habitualmente (src/main/java).
    - Con las pruebas en (src/main/test)
      - Se incluyen las dependencias Maven para
        - » Junit 5.
        - » selenium-server3.141.59.jar

Pasos para el diseño de pruebas con Selenium

- Ser sistemático etiquetando los atributos de las vistas, Ids, style, … (analizar el proyecto Notaneitor final).
- Crear un PageObject por vista o conjuntos de vistas con misma interacción.
- Diseñar y definir los casos de test:
  - Casos válidos
  - Casos inválidos

Esquema de una suite de Pruebas

```
@SpringBootTest
//Ordenamos las pruebas por la anotación @Order de cada método
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
class NotaneitorApplicationTests {
   //Para MACOSX
   static String PathFirefox = "/Applications/Firefox.app/Contents/MacOS/firefox-bin";
   static String Geckodriver = "/Users/delacal/selenium/geckodriver-v0.30.0-macos";
   //Para Windows
   //static String Geckodriver = "C:\\Path\\geckodriver-v0.30.0-win64.exe";
   //static String Geckodriver = "C:\\Dev\\tools\\selenium\\geckodriver-v0.30.0-
win64.exe";        //Común a Windows y a MACOSX
   static final String URL = "http://localhost:8090";
   static WebDriver driver = getDriver(PathFirefox, Geckodriver);

   public static WebDriver getDriver(String PathFirefox, String Geckodriver) {
      System.setProperty("webdriver.firefox.bin", PathFirefox);
      System.setProperty("webdriver.gecko.driver", Geckodriver);
      driver = new FirefoxDriver();
      return driver;
   }

   //Antes de la primera prueba
   @BeforeAll
   static public void begin() {

   }

   //Al finalizar la última prueba
   @AfterAll
   static public void end() {
      //Cerramos el navegador al finalizar las pruebas
      driver.quit();
   }

   //Antes de cada prueba se navega al URL home de la aplicación
   @BeforeEach
   public void setUp() {
      driver.navigate().to(URL);
   }

   //Después de cada prueba se borran las cookies del navegador
   @AfterEach
   public void tearDown() {
      driver.manage().deleteAllCookies();
   }    }
```

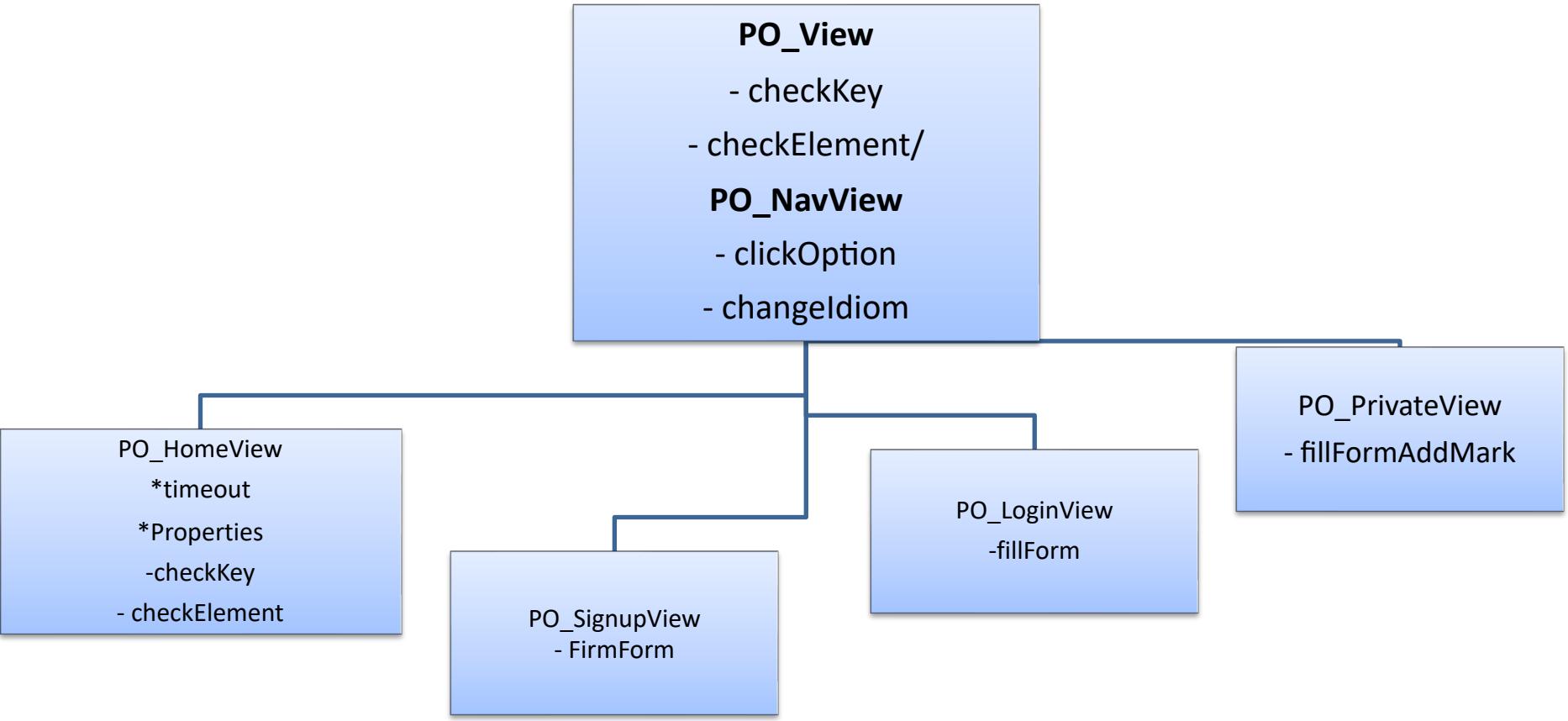
Esquema de un caso de prueba

```
@Test
@Order(1)
    public void metodo_prueba
    {
      //Paso1. Solicitud de página
        Driver.get(URL)
      //Paso2 .Esperamos carga de pagina.
        elementos = SeleniumUtils. waitLoadElementsBy(driver, tipo_elemento, cadena, timeout);
      //Paso3. Interacción con la pagina … Pinchar, rellenar, ….
      elementos.get(0).click(); // Por ejemplo
      //Paso4. Esperar por  la respuesta a la interacción
       elementos = SeleniumUtils. waitLoadElementsBy(driver, tipo_elemento, cadena, timeout);
        //Paso5. Assert de comprobacion.
      Assertrue("No se obtuvo el resultado esperado", condicion_basada_elementos);
        //Empezar en Paso 3. de Nuevo si es necesario según la prueba.
      }
```

SeleniumUtils

- Ver el código del proyecto a probar
- textIsPresentOnPage
- textIsNotPresentOnPage
- waitTextIsNotPresentOnPage
- waitLoadElementsByXpath
- waitLoadElementsBy
- waitSeconds

Propuesta de jerarquía de PO



Ver el código de la solución del proyecto a probar

Criterios de Diseño de casos de prueba

- Cada caso de prueba debe abordar sólo una entidad de datos (p.e. si se prueba el registro de un usuario, se deberá registrar un usuario por caso de prueba)
- Todos los casos deben finalizar con un aserto.

```
 ….
//Vamos al formulario de registro
    PO_HomeView.clickOption(driver, "signup", "class", "btn btn-primary");
    //Rellenamos el formulario.
    PO_SignUpView.fillForm(driver, "77777778A", "Josefo", "Perez", "77777", "77777");
    //Comprobamos que entramos en la sección privada y nos nuestra el texto a buscar
    String checkText = "Notas del usuario";
    List<WebElement> result = PO_View.checkElementBy(driver, "text", checkText);
     //CODIGO INCORRECTO. Falta un aserto aquí
}
```

Sistemas Distribuidos e Internet

Tema 4

Introducción al Web Testing

(Selenium)