Sistemas Distribuidos e Internet

Web Testing con Selenium

Sesión - 5

Curso 2022/2023

Contenido

Contenido 2
1 Introducción 4
1.1 Modificación del proyecto que vamos a probar 4
2 Desarrollo de las pruebas para Notaneitor 5
2.1 Selección Selenium y del navegador de pruebas 5
2.2 Ubicación y lanzamiento del proyecto a probar (Notaneitorv3.0) 6
2.3 Diseño e implementación de las pruebas (NotaneitorTests) 7
2.3.1 Elección del framework de test 7
2.3.2 Creación de la batería de pruebas JUnit sobre un proyecto Spring Boot y ejecución 7
2.3.2.1 Librerías necesarias 7
2.3.2.2 Ejecución de dos proyectos en uno 8
2.3.2.3 Desarrollo de las pruebas 8
2.3.2.3.1 Clase principal Junit 8
2.3.2.3.2 Creación de los casos de prueba(test) 9
2.3.2.3.3 Anotación de la clase principal para ejecución ordenada 10
2.3.2.3.4 Diseño de los casos de prueba y uso de la Consola de Firefox 11
2.3.2.4 Diseño de las clases 12
2.3.2.4.1 Clase base PO_View 12
2.3.2.4.2 Clase base NavView 13
2.3.2.4.3 PO_HomeView y casos de prueba para la vista Home 15
2.3.2.4.3.1 PR01: Acceso a la página principal 17
2.3.2.4.3.2 PR02: Ir la vista de Registro, PR03: Ir a la vista de Login 17
2.3.2.4.3.3 PR04: Botones de idioma 18
2.3.2.4.4 PO_SignUpView y casos de pruebas para la vista Registrarse 18
2.3.2.4.4.1 PR05 y PR06: Registro de usuario 19
2.3.2.4.5 PO_LoginView (PR07-11) 20
2.3.2.4.6 PO_PrivateView: Vista privada de estudiante/profesor 21
2.3.2.4.6.1 PR12: Lista de Notas 22
2.3.2.4.6.2 PR13: Detalle de una Nota 23

2.3.2.4.6.3 PR14: Agregar una nota 24
2.3.2.4.6.4 PR15: Eliminar una nota 25
2.4 Cuestiones generales 26
2.4.1 Generación del archivo JAR para despliegue 26
2.4.2 Ejecución de las pruebas varias veces 27
2.4.3 Refactorización de código 27
2.5 Etiquetar proyecto en GitHub 27
2.6 Resultado esperado en el repositorio de GitHub 27

1 Introducción

Una de las mayores inversiones en la industria del software radica en el **mantenimiento** del mismo. Y dentro de esas cuantiosas inversiones está la prueba sistemática. Porque realmente cuando hablamos de prueba de software no hablamos exclusivamente de pruebas alpha y beta de los entregables al cliente, sino de también de las pruebas en los cambios una vez el software está implantado.

Es por ello por lo que son necesarias herramientas que faciliten al equipo desarrollador sistematizar este tipo de tareas. Dentro del mundo del desarrollo web, la prueba es un proceso un poco más complejo que en el software de backend ya que el código está siempre muy acoplado a la herramienta de desarrollo (cliente web y servidor web). Por lo tanto, se emplean herramientas específicas que nos facilitan esa tarea, como es el caso del framework Selenium.

En el Web Testing sólo se prueba la parte de la capa de presentación renderizada en al navegador, asumiendo que las capas inferiores ya han sido probadas. Para la prueba de las capas inferiores hay otro tipo de técnicas más sencillas que no son objetivo de esta asignatura.

Prerrequisitos / Software utilizado:

1. **Selenium V.3.141.59**: Entorno de pruebas para aplicaciones web.
2. **Geckodriver-v0.30.0:** Driver para el navegador Firefox.
3. **Firefox 97.0 o superior (64-bit):** Navegador web Firefox.
4. **Notaneitor-0.0.1-SNAPSHOT.jar**: Aplicación web a probar.
5. **Junit-jupiter versión 5.8.**1: Librerías de pruebas unitarias de aplicaciones Java.
6. **JDK-17.0.1:** Java Development Kit.

1.1 Modificación del proyecto que vamos a probar

Vamos a probar el proyecto Spring Boot creado en la sesión anterior tratando de respetar el código HTML generado por Thymeleaf. ¿Por qué lo vamos a hacer así? Pues porque en la vida real muchas veces no tenemos opción de generar el código HTML como nosotros quisiéramos. Por ello, vamos a intentar adaptarnos al proyecto tal cual lo tenemos en términos de etiquetado, respetando los textos, archivos de propiedades, atributos de elementos html ("id", "name", …). O**bviamente si añadiéramos "ids" y clases ("class") a todos los elementos HTML necesarios, las cosas serían más fáciles**, **pero vamos a dejarlo tal cual está**.

Notas: Para el caso de la práctica entregable, **SI** deberían generarse con los ids y clases deseables con el fin de hacer más fácil el código de Selenium.

¡¡¡¡¡MUY IMPORTANTE!!!!!!

Como esta práctica guiada es una continuación de la anterior, seguiremos utilizando el mismo repositorio de Github de la práctica anterior.

2 Desarrollo de las pruebas para Notaneitor

Los pasos que se detallarán en esta sección son válidos tanto para **MACOSX** como **Windows**. En el caso de que hubiera alguna diferencia se indicará mediante los comentarios oportunos en los puntos concretos.

Por otro lado, hay que decir que la metodología de trabajo que se va a explicar en este apartado es válida para cualquier tipo de aplicación web, independientemente de la tecnología que se haya empleado para su desarrollo. Desde una página web estática desarrollada a mano, hasta una aplicación web desarrollada con tecnología de cliente como pueda ser Angular, React, Vue, etc. También sería válida para aplicaciones web de servidor empleando frameworks como PrimeFaces que generan mucho código JavaScript bastante **críptico**.

Los pasos que se deben seguir para desarrollar una batería de pruebas Selenium para una aplicación Web serán los siguientes:

1. Selección de la versión de Selenium, así como el navegador de pruebas, en nuestro caso Selenium 3 y Firefox 97.0
2. Ubicación y lanzamiento del proyecto a probar, en nuestro caso, Notaneitor.
3. Diseño e implementación de las pruebas, en nuestro caso NotaneitorTests.

2.1 Selección Selenium y del navegador de pruebas

Actualmente cuando nos planteamos testear con Selenium tenemos que tomar una doble decisión: por un lado, decidir qué versión de Selenium emplearemos y por otro el navegador con el que vamos a ejecutar las pruebas.

En cuanto a la versión de Selenium, podemos decir que la configuración de Selenium2 es muy simple ya que incorpora en la propia librería los drivers para los navegadores más populares, mientras que Selenium 3 y 4 y exigen instalar un driver especifico según la subversión x de Selenium3.x y la versión de navegador, además de una pequeña configuración. Por ejemplo, para Selenium 3.141/Firefox 64 o superior se debe instalar geckodriver 0.24 o superior.

Respecto al navegador, podemos decir que tanto en la versión 2, 3 y 4 de Selenium disponemos de soporte para los navegadores más populares: Firefox, Chrome, Edge y Safari. En esta práctica usaremos la Selenium 3 porque la versión 4 aun no está muy madura y tiene bastantes fallos.

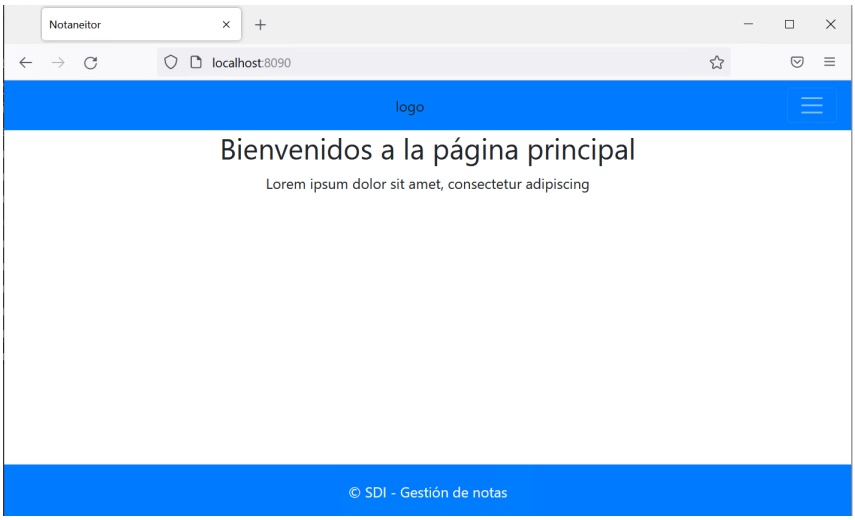
NOTA: En nuestro caso vamos a optar por irnos a la versión de **Firefox 97.0** o superior lo cual nos implica el uso de Selenium 3 (**Selenium 3.141**) y el driver **geckodrive 0.30.0**.

2.2 Ubicación y lanzamiento del proyecto a probar

En este caso vamos a probar una solución parcial obtenida a partir de la última sesión de prácticas con Springboot que se ha suministrado en con el material de esta práctica en su versión ejecutable **(notaneitor-0.0.1-SNAPSHOT.jar)**.

Para ejecutar la aplicación web a probar, sigue los siguientes pasos:

1. Primero **no olvides lanzar la base de datos**.
2. Abre la consola de tu sistema operativo.
3. A continuación, sitúate en la carpeta donde has descomprimido el material y ejecuta el jar del proyecto, por la línea de comando:

   $> java -jar notaneitor-0.0.1-SNAPSHOT.jar
4. Ya puedes probar el proyecto suministrado con Firefox 97.0, desde el siguiente enlace: http://localhost:8090
5. 

ACLARACIÓN IMPORTANTE: Aunque las pruebas que vamos a desarrollar las vas a incluir en tu proyecto Spring Boot (más adelante veremos cómo), **el proyecto que ejecutaremos para probar será el que te hemos suministrado en formato .jar y no el tuyo**. **Esto es debido a que los nombres de los campos y los diferentes elementos HTML de las vistas puede que sean distintos de los tuyos**. De esta forma aseguramos que el código que vamos a desarrollar funcione correctamente.

2.3 Diseño e implementación de las pruebas (NotaneitorTests)

2.3.1 Elección del framework de test

Los dos frameworks de test unitarios basados en java más populares hoy en día son JUnit y TestNG. TestNG es por decirlo de alguna manera es una mejora de JUnit 4, ya que, aunque está basado en el mismo esquema de asertos, incorpora anotaciones más potentes como puede ser la siguiente:

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

Estas anotaciones permiten ejecución multihilo en paralelo de una prueba. No obstante, para el propósito de esta sesión es suficiente JUnit. Esto añadido a que es una herramienta que ya conocemos.

En esta práctica emplearemos como framework de prueba **JUNIT,** ya que se trata de una herramienta ya conocida.

2.3.2 Creación de la batería de pruebas JUnit sobre un proyecto Spring Boot y ejecución

Con el fin de facilitar las cosas vamos a crear las pruebas sobre el proyecto Spring Boot Notaneitor que ha sido desarrollando durante estas semanas en IntelliJ. Para ello debemos seguir los siguientes pasos:

1) Incluir las librerías necesarias Selenium y JUnit5 en el proyecto.
2) **Desarrollar las clases necesarias para las pruebas sobre la carpeta de código de pruebas (src/test/java)**.
3) Ejecutar las pruebas.

2.3.2.1 Librerías necesarias

Antes de comenzar a desarrollar incorporaremos las dos librerías necesarias en nuestro proyecto Spring Boot para desarrollar nuestra suite de pruebas JUnit/Selenium. La librería de Selenium se puede añadir al BuildPath como un fichero jar o directamente desde POM, si es un proyecto Maven, como es nuestro caso.

- En el proyecto Springboot desarrollado en clase, añadiremos al POM las dependencias de Selenium y de JUNIT5. En caso de JUNIT no es obligatorio cuando usamos el IDE Intellij porque por defecto viene configurado con la última versión de JUNIT5.

Aunque, en nuestro caso, vamos a especificar la versión utilizada para garantizar que lo desarrollado funciona correctamente.

```
<dependency>
  <groupId>org.seleniumhq.selenium</groupId>
  <artifactId>selenium-java</artifactId>
  <version>3.141.59</version>
</dependency>

<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter</artifactId>
  <version>5.8.1</version>
</dependency>
```

De esta forma ya tenemos el proyecto preparado para desarrollar con JUnit/Selenium.

2.3.2.2 Ejecución de dos proyectos en uno

Para ejecutar cada proyecto debemos seleccionar la opción correspondiente:

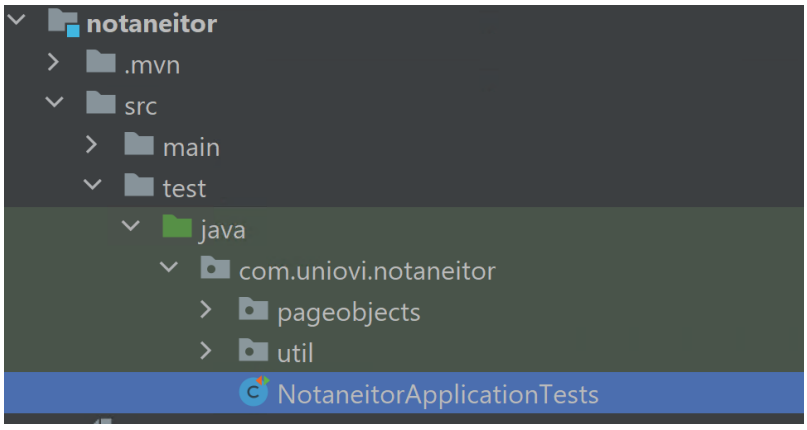
- Para el proyecto Spring Boot seleccionaremos sobre el menú contextual del proyecto la opción **Run 'NotaneitorAplicacion'**. O Run NOMBREDELAAPLICACIÓN
- Para el proyecto JUnit seleccionaremos sobre el menú contextual del proyecto la opción **Run 'NotaneitorApplicationTests"**

2.3.2.3 Desarrollo de las pruebas

A continuación, vamos a desarrollar paso a paso algunas pruebas.

2.3.2.3.1 Clase principal Junit

En la clase principal de prueba del proyecto Springboot (NotaneitorApplicationTests), copiamos el código:



```
static String PathFirefox = "C:\\Program Files\\Mozilla Firefox\\firefox.exe";
//static String Geckodriver = "C:\\Path\\geckodriver-v0.30.0-win64.exe";
static String Geckodriver = "C:\\Dev\\tools\\selenium\\geckodriver-v0.30.0-win64.exe";

//static String PathFirefox = "/Applications/Firefox.app/Contents/MacOS/firefox-bin";
//static String Geckodriver = "/Users/USUARIO/selenium/geckodriver-v0.30.0-macos";

//Común a Windows y a MACOSX
static WebDriver driver = getDriver(PathFirefox, Geckodriver);
static String URL = "http://localhost:8090";

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;
}
```

Sobre este código haremos dos cosas:

- Incluir los datos miembro necesarios para las pruebas.

NOTA: Debes descomentar y modificar las líneas correspondientes a PathFirefox y Geckodriver que se correspondan a tu SO y modificar el valor del path para que apunte a donde tengas el ejecutable de Firefox, así como el driver de selenium geckodriver (en el material suministrado se adjunta el driver gecko tanto para Mac como para Windows. Debes emplear el que corresponda).

- Modificar los métodos @BeforeAll/AfterAll/BeforeEach/AfterEach/ para que el contexto de partida en todos los casos de prueba sea siempre el mismo.

```
@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();
}
//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();
}
```

2.3.2.3.2 Creación de los casos de prueba(test)

La batería de pruebas se va a realizar siguiendo los siguientes pasos:

1) Copia la utilidad de test suministrada con el material de la sesión (***SeleniumUtils.java***) al paquete ***com.uniovi.notaneitor.util*** (**¡Recuerda! Estamos en src/test/java**). Esta utilidad es una clase con métodos estáticos basados en Selenium. Gracias a estos métodos, se puede implementar la mayoría de las comprobaciones de tus pruebas, sin necesidad de tener que recurrir directamente a los métodos nativos de Selenium. Esta utilidad está suficientemente documentada, por lo que no se explicará aquí salvo cuando se haga uso de ella.

2) Anotar la clase principal ***NotaneitorApplicationTests*** para que las pruebas se ejecuten de forma ordenada creciente según el nombre del método de prueba (***@TestMethodOrder(MethodOrderer.OrderAnnotation.class***)).

3) Diseñar el código necesario para cada prueba y anotar los test con la anotación ***@Order(N)***. Donde N es el numero de orden de ejecución. Por ejemplo, ***@Order(1)***.

2.3.2.3.3 Anotación de la clase principal para ejecución ordenada

Con el fin de que los casos de prueba se ejecuten siempre en el mismo orden se debe incluir la anotación siguiente para la clase NotaneitorTests:

```
@SpringBootTest
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
class NotaneitorApplicationTests {
   static String PathFirefox = "C:\\Program Files\\Mozilla Firefox\\firefox.exe";
 ...
}
```

Y por lo tanto si creamos 10 pruebas y se ejecutan con éxito las 10, tendremos el siguiente resultado:

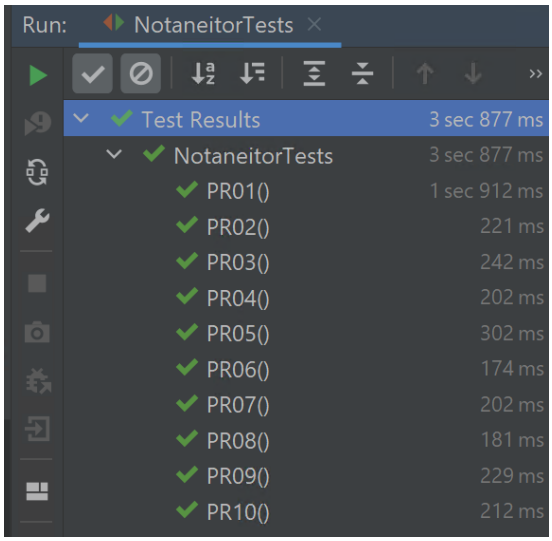
```
@Test
@Order(1)
void PR01() {}

@Test
@Order(2)
void PR02() {}

@Test
@Order(3)
void PR03() {}

@Test
@Order(4)
void PR04() {}

@Test
@Order(5)
void PR05() {}
```



De otra forma JUnit no ejecutará siguiendo un orden determinista los casos de prueba.

Nota: Incluir el siguiente Commit Message ->

"SDI-IDGIT-5.1-Clase principal."

OJO: sustituir IDGIT por tu número asignado (p.e. 2223-101):

"SDI-2223-101-5.1-Clase principal."

(No olvides incluir los guiones y NO incluyas BLANCOS)

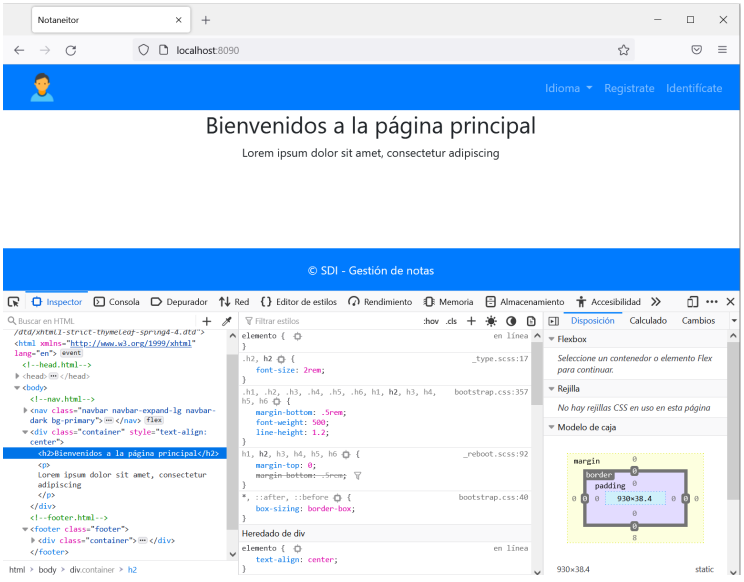
2.3.2.3.4 Diseño de los casos de prueba y uso de la Consola de Firefox

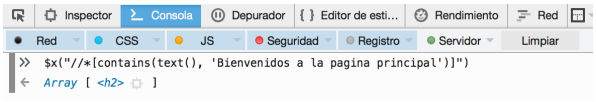
El diseño de casos de pruebas mediante Selenium está basado en **verificar** la interacción que haría un usuario humano con el caso de uso que se pretende probar. Cuando decimos verificar nos referimos a comprobar que, tras cada interacción, el contenido de la página siguiente es el que debe ser. A modo de ejemplo, se incluye el siguiente fragmento de código donde se comprueba el cambio del idioma en el mensaje de saludo principal de Notaneitor.

```
...
   //Cambiamos el idioma a Inglés
  PO_HomeView.changeLanguaje (driver, "btnEnglish"));
  //Esperamos porque aparezca que aparezca el texto de bienvenida en inglés
SeleniumUtils. waitLoadElementsBy(driver, "text", p.getString("welcome.message",  PO_Properties.ENGLISH), getTimeout());
...
```

Normalmente, no se comprueba todo el contenido ya que sería un proceso poco óptimo, sino que se comprueban elementos que consideramos clave (un enlace determinado, un texto que debe aparecer o desaparecer, el contenido de un campo, …).

Por otro lado, cuando se implementa la búsqueda de un elemento es necesario diseñar la consulta xpath, que suele ser sencilla de definir, pero otras veces no tanto. Por ello se recomienda inicialmente usar el Inspector de Firefox (***Herramientas/Desarrollador Web/Inspector***) para identificar el elemento que queremos seleccionar, para a continuación probar la consulta xpath mediante la consola del mismo Firefox (***Herramientas/Desarrollador Web/Consola***).



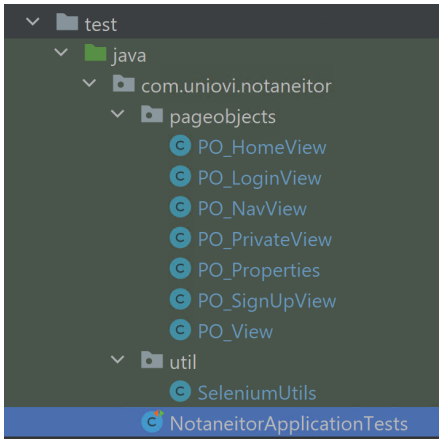


Haciendo ***Click derecho-> copiar->xpath*** sobre un elemento HTML obtendremos la ruta completa de ese elemento.

Con el fin de poder reutilizar nuestro código, así como hacerlo más mantenible vamos a crear una jerarquía de clases que se describirá en la siguiente sección:

2.3.2.4 Diseño de las clases

Para probar cada vista se empleará un patrón PageObject (PO) para cada vista o conjunto de vistas con similar interacción. El patrón PO envuelve los métodos de prueba relativos a la vista que representa (Home, Login, Register, …). Durante esta sesión crearemos los POs que se muestran en la siguiente figura:



Vamos a crear dos clases base con propiedades y métodos generales para los POs que vayamos creando:

- PO_View: Contiene las propiedades comunes a todos los POs.
- PO_NavView: Deriva de PO_View y será de la que hereden el resto de POs.

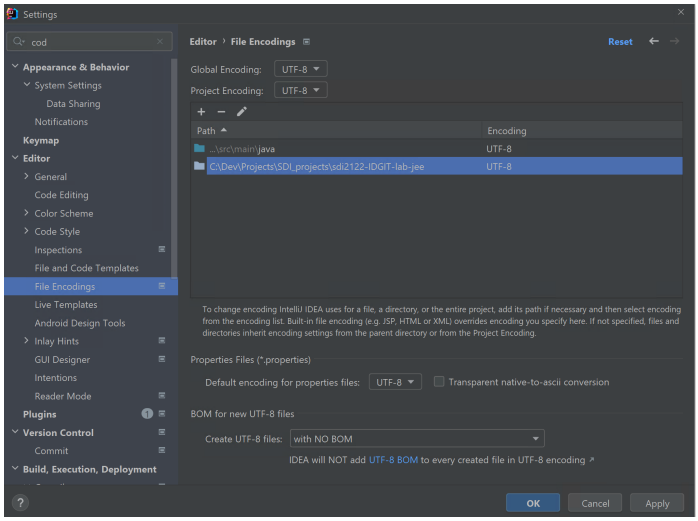
2.3.2.4.1 Clase base PO_View

PO_View contiene las propiedades comunes a todos los PO:

- int timeout: el tiempo de espera que se empleará para cada búsqueda de un elemento.
- PO_Properties p: un envoltorio para los archivos de propiedades empleados en el proyecto web a probar. No es habitual disponer de este estos datos y, en caso de no disponer de ellos, se pueden crear a partir de la propia web disponible con el fin de facilitar la implementación de las pruebas. En nuestro caso ya disponemos de estos datos.

Te suministramos esta clase además de PO_Properties en el material de la sesión:

- Crea el paquete ***com.uniovi.notaneitor.pageobjects.***
- Copia el archivo ***PO_View.java*** al paquete ***com.uniovi.notaneitor.pageobjects*** de la carpeta src/test/java.
- Copia el archivo ***PO_Properties.java*** al paquete ***com.uniovi.notaneitor.pageobjects*** de la carpeta src/test/java.
- Copia también los archivos de propiedades que te suministramos en el material de esta práctica a la carpeta ***src/test/java*** y en la carpeta ***src/main/java/resources***
- Asegúrate que todo el proyecto este configurado con el encoding en UTF8. Esto puedes hacerlo desde el menú **File|Setting** del Intellij y busca Editor|File Encoding. Tal como se muestra en la siguiente imagen:



Nota: Incluir el siguiente Commit Message ->

"SDI-IDGIT-5.2-Clases Auxiliares Base."

OJO: sustituir IDGIT por tu número asignado (p.e. 2223-101):

"SDI-2223-101-5.2-Clases Auxiliares Base."

(No olvides incluir los guiones y NO incluyas BLANCOS)

2.3.2.4.2 Clase base NavView

Dado que todas nuestras vistas disponen un menú de navegación (con más o menos opciones según el rol), vamos a crear un PO para las opciones de navegación denominado PO_NavView, que heredará de PO_View. El resto de POs correspondientes a las diferentes vistas heredarán de PO_NavView. Crear el archivo ***PO_NavView.java*** a partir del siguiente código:

```
package com.uniovi.notaneitor.pageobjects;
```

```
import com.uniovi.notaneitor.util.SeleniumUtils;
import org.junit.jupiter.api.Assertions;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import java.util.List;

public class PO_NavView  extends PO_View{
}
```

En esta clase vamos a incorporar dos métodos:

- clickOption: para probar cualquiera de las opciones principales del menú.
- changeLanguage: para probar las opciones de cambio de idioma.

Agrega a la clase PO_NavView el código de clickOption:

```
/**
 * Clic en una de las opciones principales (a href) y comprueba que se vaya a la vista con el elemento de
tipo type con el texto Destino
 * @param driver: apuntando al navegador abierto actualmente.
 * @param textOption: Texto de la opción principal.
 * @param criterio: "id" or "class" or "text" or "@attribute" or "free". Si el valor de criterio es free es una
expresion xpath completa.
 * @param targetText: texto correspondiente a la búsqueda de la página destino.
 */
public static void clickOption(WebDriver driver, String textOption, String criterio, String targetText) {
   //CLickamos en la opción de registro y esperamos a que se cargue el enlace de Registro.
   List<WebElement> elements = SeleniumUtils.waitLoadElementsBy(driver, "@href", textOption,
getTimeout());
   //Tiene que haber un sólo elemento.
   Assertions.assertEquals(1, elements.size());
   //Ahora lo clickamos
   elements.get(0).click();
   //Esperamos a que sea visible un elemento concreto
   elements = SeleniumUtils.waitLoadElementsBy(driver, criterio, targetText, getTimeout());
   //Tiene que haber un sólo elemento.
   Assertions.assertEquals(1, elements.size());
}
```

Este método nos permite indicarle a Selenium que haga clic en un enlace con texto textOption (***waitLoadElementsBy (driver, "@href….)*** y que espere que se cargue otro elemento según la consulta xpath criterio/targetText (**waitLoadElementsBy(driver, criterio,**). Lo probaremos con las vistas Login y Register. Ahora copia el método changeLanguage:

```
/**
 * Selecciona el enlace de idioma correspondiente al texto textLanguage
```

```
 * @param driver: apuntando al navegador abierto actualmente.
 * @param textLanguage: el texto que aparece en el enlace de idioma ("English" o "Spanish")
 */
public static void changeLanguage(WebDriver driver, String textLanguage) {
   //clickamos la opción Idioma.
   List<WebElement> languageButton = SeleniumUtils.waitLoadElementsBy(driver, "id", "btnLanguage",
getTimeout());
   languageButton.get(0).click();
   //Esperamos a que aparezca el menú de opciones.
   SeleniumUtils.waitLoadElementsBy(driver, "id", "languageDropdownMenuButton", getTimeout());
   //CLickamos la opción Inglés partiendo de la opción Español
   List<WebElement> Selectedlanguage = SeleniumUtils.waitLoadElementsBy(driver, "id", textLanguage,
getTimeout());
   Selectedlanguage.get(0).click();
}
```

Este método despliega el menú de idioma haciendo clic en el desplegable con id="btnLanguage" (***waitLoadElementsBy (driver, "id", "btnLanguage",)*** y después selecciona el idioma cuyo id debe ser textLanguage (***waitLoadElementsBy (driver, "id", textLanguage***). Fíjate que tras pinchar la opción btnLanguage debes esperar a que aparezca el menú desplegable con los idiomas (***waitLoadElementsBy(driver, "id", "languageDropdownMenuButton***).

De esta forma dejamos cerrada la clase PO_NavView.

MODIFICACIONES DEL CODIGO PROPUESTO

Por supuesto que esta y todas las clases están abiertas a cuantas modificaciones se deseen realizar. Es importante comentar con el profesor estos cambios.

Nota: Incluir el siguiente Commit Message ->

"SDI-IDGIT-5.3-PO NavView."

OJO: sustituir IDGIT por tu número asignado (p.e. 2223-101):

"SDI-2223-101-5.3-PO NavView."

(No olvides incluir los guiones y NO incluyas BLANCOS)

Una vez tenemos creadas las clases auxiliares vamos a comenzar a probar las diferentes vistas.

2.3.2.4.3 PO_HomeView y casos de prueba para la vista Home

Para probar la página Home vamos a crear el PO PO_HomeView.java que herede de PO_NavView, en la que vamos a incluir dos métodos:

- checkWelcomeToPage: Para comprobar el mensaje de bienvenida en la página Home.
- checkChangeLanguage: Para comprobar la interacción con los botones de cambio de idioma

Método checkWelcomeToPage y getWelcomeMessageText

Incluye el texto del método checkWelcomeToPage:

```
static public void checkWelcomeToPage(WebDriver driver, int language) {
   //Esperamos a que se cargue el saludo de bienvenida en Español
   SeleniumUtils.waitLoadElementsBy(driver, "text", p.getString("welcome.message", language),
getTimeout());
}
```

Este método está pensado exclusivamente para comprobar el mensaje de bienvenida, pero podría generalizarse para buscar cualquier clave del archivo de propiedades. **Dejamos ese tema pendiente si es que lo crees necesario para tu práctica**.

Otra forma de verificar una condición es devolver el elemento HTML y que la validación del resultado se realice en el test con la correspondiente Assertion. Esta es otra forma de hacer este método:

```
static public List<WebElement> getWelcomeMessageText(WebDriver driver, int language) {
   //Esperamos a que se cargue el saludo de bienvenida en Español
   return SeleniumUtils.waitLoadElementsBy(driver, "text", p.getString("welcome.message", language),
getTimeout());
}
```

Método checkChangeLanguage

Incluye ahora el código para el método checkChangeLanguage:

```
static public void checkChangeLanguage(WebDriver driver, String textLanguage1, String textLanguage,
int locale1, int locale2) {
   //Esperamos a que se cargue el saludo de bienvenida en Español
   PO_HomeView.checkWelcomeToPage(driver, locale1);
   //Cambiamos a segundo idioma
   PO_HomeView.changeLanguage(driver, textLanguage);
   //Comprobamos que el texto de bienvenida haya cambiado a segundo idioma
   PO_HomeView.checkWelcomeToPage(driver, locale2);
   //Volvemos a Español.
   PO_HomeView.changeLanguage(driver, textLanguage1);
   //Esperamos a que se cargue el saludo de bienvenida en Español
   PO_HomeView.checkWelcomeToPage(driver, locale1);
}
```

En este método se aprovechan los métodos ya creados: PO_NavView.changeLanguage y PO_HomeView.checkWelcomePage para comprobar que funcionan correctamente los cambios de idioma.

A continuación, vamos a desarrollar 4 casos de prueba para la vista Home:

1. PR01: Acceso a la página principal
2. PR02: Ir al formulario de Registro.
3. PR03: Ir al formulario de Login.
4. PR04: Cambiar el idioma.

2.3.2.4.3.1 PR01: Acceso a la página principal

En la clase ***NotaneitorApplicationTests*** incluimos todos los test de la aplicación. Esta prueba queda muy fácil, sólo tienes definir el método correspondiente, anotar la clase con @Test y luego que incluir la llamada a PO_HomeView.checkWelcome :

```
@Test
@Order(1)
void PR01A() {
   PO_HomeView.checkWelcomeToPage(driver, PO_Properties.getSPANISH());
}

@Test
@Order(2)
void PR01B() {
   List<WebElement> welcomeMessageElement = PO_HomeView.getWelcomeMessageText(driver,
PO_Properties.getSPANISH());
   Assertions.assertEquals(welcomeMessageElement.get(0).getText(),
PO_HomeView.getP().getString("welcome.message", PO_Properties.getSPANISH()));
}
```

En este caso, la prueba ***PR01A()*** llama el método ***checkWelcomeToPage()*** que ya tiene su propio aserto. En la prueba ***PR01B()*** llama el método ***getWelcomeMessageText()*** que devolverá o no un elemento HTML, por lo que se ha añadido un aserto a la prueba, comprobando que todo está correcto. Esta última forma de prueba es más clara y recomendable, porque hacemos la validación en la propia prueba.

Para probar los test recuerda que tienes que tener en ejecución HSQLDB y el Jar proporcionado. Ahora, Ejecuta las pruebas situando el punto del ratón sobre la cabecera del método y en el menú contextual seleccionado "Run NotaneitorAplicationTest":

2.3.2.4.3.2 PR02: Ir la vista de Registro, PR03: Ir a la vista de Login

En estos dos casos sólo comprobaremos la navegación desde la página Home a las vistas de Registro y Login respectivamente. Emplearemos el método PO_HomeView.clickOption.

Debes **conocer el valor atributo id del enlace** de cada opción de menú: id="signup" e id="login" **y el valor del atributo class** que aparece en los formularios de Registro y Login (class="btn btn-primary"). El propio método clickOption se encarga de esperar de realizar la navegación.

Copia el código de ambos tests y pruébalos:

```
//PR02. Opción de navegación. Pinchar en el enlace Registro en la página home
@Test
@Order(3)
public void PR02() {
   PO_HomeView.clickOption(driver, "signup", "class", "btn btn-primary");
}

//PR03. Opción de navegación. Pinchar en el enlace Identifícate en la página home
@Test
@Order(4)
public void PR03() {
   PO_HomeView.clickOption(driver, "login", "class", "btn btn-primary");
}
```

2.3.2.4.3.3 PR04: Botones de idioma

Para cambiar el idioma emplearemos el método PO_HomeView.checkChangeLanguaje. Copia y ejecuta el siguiente código.

```
//PR04. Opción de navegación. Cambio de idioma de Español a Inglés y vuelta a Español
@Test
@Order(5)
public void PR04() {
   PO_HomeView.checkChangeLanguage(driver, "btnSpanish", "btnEnglish",
PO_Properties.getSPANISH(), PO_Properties.getENGLISH());
}
```

Nota: Incluir el siguiente Commit Message ->

"SDI-IDGIT-5.4-Casos Uso VistaHome."

OJO: sustituir IDGIT por tu número asignado (p.e. 2223-101):

"SDI-2223-101-5.4-Casos Uso VistaHome."

(No olvides incluir los guiones y NO incluyas BLANCOS)

2.3.2.4.4 PO_SignUpView y casos de pruebas para la vista Registrarse

Para los casos de prueba del formulario de registro crearemos la clase PO_SignUpView con el método fillForm:

```
public class PO_SignUpView extends PO_NavView {
   static public void fillForm(WebDriver driver, String dnip, String namep, String lastnamep, String
passwordp, String passwordconfp) {
      WebElement dni = driver.findElement(By.name("dni"));
      dni.click();
      dni.clear();
      dni.sendKeys(dnip);
```

```
    WebElement name = driver.findElement(By.name("name"));
    name.click();
    name.clear();
    name.sendKeys(namep);
    WebElement lastname = driver.findElement(By.name("lastName"));
    lastname.click();
    lastname.clear();
    lastname.sendKeys(lastnamep);
    WebElement password = driver.findElement(By.name("password"));
    password.click();
    password.clear();
    password.sendKeys(passwordp);
    WebElement passwordConfirm = driver.findElement(By.name("passwordConfirm"));
    passwordConfirm.click();
    passwordConfirm.clear();
    passwordConfirm.sendKeys(passwordconfp);
    //Pulsar el boton de Alta.
    By boton = By.className("btn");
    driver.findElement(boton).click();
  }
}
```

2.3.2.4.4.1 PR05 y PR06: Registro de usuario

Vamos a definir un caso de test válido (PR05) para esta vista y un caso de test inválido (PR06).

Copia el código para el caso de test PR05:

```
//PR05. Prueba del formulario de registro. registro con datos correctos
@Test
@Order(6)
public void PR05() {
  //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);
  Assertions.assertEquals(checkText, result.get(0).getText());
}
```

En este caso vamos al formulario de registro, lo rellenamos y esperamos a visualizar la cabecera de la lista de notas para ese alumno (“Notas del usuario”).

Copia el caso de prueba inválido (PR06A y PR06B) para el formulario de registro:

```
//PR06A. Prueba del formulario de registro. DNI repetido en la BD
// Propiedad: Error.signup.dni.duplicate
@Test
@Order(7)
public void PR06A() {
```

```
    PO_HomeView.clickOption(driver, "signup", "class", "btn btn-primary");
    PO_SignUpView.fillForm(driver, "99999990A", "Josefo", "Perez", "77777", "77777");
    List<WebElement> result = PO_SignUpView.checkElementByKey(driver, "Error.signup.dni.duplicate",
PO_Properties.getSPANISH() );
    //Comprobamos el error de DNI repetido.
    String checkText = PO_HomeView.getP().getString("Error.signup.dni.duplicate",
PO_Properties.getSPANISH());
    Assertions.assertEquals(checkText , result.get(0).getText());
}
//PR06B. Prueba del formulario de registro. Nombre corto.
// Propiedad: Error.signup.dni.length
@Test
@Order(8)
public void PR06B() {
    PO_HomeView.clickOption(driver, "signup", "class", "btn btn-primary");
    PO_SignUpView.fillForm(driver, "99999990B", "Jose", "Perez", "77777", "77777");
    List<WebElement> result =  PO_SignUpView.checkElementByKey(driver, "Error.signup.name.length",
PO_Properties.getSPANISH() );
    //Comprobamos el error de Nombre corto de nombre corto .
    String checkText = PO_HomeView.getP().getString("Error.signup.name.length",
PO_Properties.getSPANISH());
```

Nota: Dejamos para ti implementar pruebas para comprobar "errores" en el resto de los campos. Puedes enumerar los casos de pruebas que desarrolles como PR06C, PR06D, etcétera.

```
    Assertions.assertEquals(checkText , result.get(0).getText());
}
```

Ten en cuenta que el test PR05 altera la base de datos, dando error en sucesivas ejecuciones, ya que no reinicializamos la BD entre tests.

Nota: Incluir el siguiente Commit Message ->

"SDI-IDGIT-5.5-Casos Uso VistaSignUp."

OJO: sustituir IDGIT por tu número asignado (p.e. 2223-101):

"SDI-2223-101-5.5-Casos Uso VistaSignUp."

(No olvides incluir los guiones y NO incluyas BLANCOS)

2.3.2.4.5 PO_LoginView (PR07-11)

Nota: Te dejamos para ti la creación del PO PO_LoginView. Inspírate en PO_RegisterView.

Después de crear el PO_LoginView, deberás crear las siguientes pruebas:

- PR07: Identificación válida con usuario de ROL usuario (99999990A/123456).
- PR08: Identificación válida con usuario de ROL profesor (99999993D/123456).

- PR09: Identificación válida con usuario de ROL Administrador (99999988F/123456).
- PR10: Identificación inválida con usuario de ROL alumno (99999990A/123456).
- PR11: Identificación válida y desconexión con usuario de ROL usuario (99999990A/123456).

Te suministramos el código del caso de prueba PR07 para que te sirva de guía:

```
@Test
@Order(9)
public void PR07() {
   //Vamos al formulario de logueo.
   PO_HomeView.clickOption(driver, "login", "class", "btn btn-primary");
   //Rellenamos el formulario
   PO_LoginView.fillLoginForm(driver, "99999990A", "123456");
   //Comprobamos que entramos en la pagina privada de Alumno
   String checkText =  "Notas del usuario";
   List<WebElement> result = PO_View.checkElementBy(driver, "text", checkText);
   Assertions.assertEquals(checkText, result.get(0).getText());
}
```

Nota: Antes de hacer el commit debes crear la clase PO_LoginView, el método fillForm y los casos de prueba ya descritos: PR8-11

Nota: Incluir el siguiente Commit Message ->

"SDI-IDGIT-5.6-Casos Uso VistaLogin."

OJO: sustituir IDGIT por tu número asignado (p.e. 2223-101):

"SDI-2223-101-5.6-Casos Uso VistaLogin."

(No olvides incluir los guiones y NO incluyas BLANCOS)

2.3.2.4.6 PO_PrivateView: Vista privada de estudiante/profesor

Para las vistas privadas de estudiante y profesor vamos a crear un PO denominado PO_PrivateView. Crear la clase correspondiente, copia y analiza el siguiente código:

```
package com.uniovi.notaneitor.pageobjects;

import com.uniovi.notaneitor.SeleniumUtils;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.ui.Select;

public class PO_PrivateView extends PO_NavView {
```

```
  static public void fillFormAddMark(WebDriver driver, int userOrder, String descriptionp, String scorep)
{
    //Esperamos 5 segundo a que carge el DOM porque en algunos equipos falla
    SeleniumUtils.waitSeconds(driver, 5);
    //Seleccionamos el alumnos userOrder
    new Select(driver.findElement(By.id("user"))).selectByIndex(userOrder);
    //Rellenemos el campo de descripción
    WebElement description = driver.findElement(By.name("description"));
    description.clear();
    description.sendKeys(descriptionp);
    WebElement score = driver.findElement(By.name("score"));
    score.click();
    score.clear();
    score.sendKeys(scorep);
    By boton = By.className("btn");
    driver.findElement(boton).click();
  }
}
```

Este PO sólo incorpora el método fillFormAddMark empleado para rellenar el formulario correspondiente a nuevas calificaciones.

Los casos de pruebas que implementaremos para esta vista son:

- PR12: Identificarse como estudiante, comprobar la lista de notas y logout.
- PR13: Identificarse como estudiante y pinchar el detalle de una nota y logout.
- PR14: Identificarse como estudiante, agregar una nota y logout.
- PR15: Identificarse como estudiante, Ir a la última página de notas, eliminar una nota y logout.

2.3.2.4.6.1 PR12: Lista de Notas

Copia a continuación el código del caso de prueba PR12:

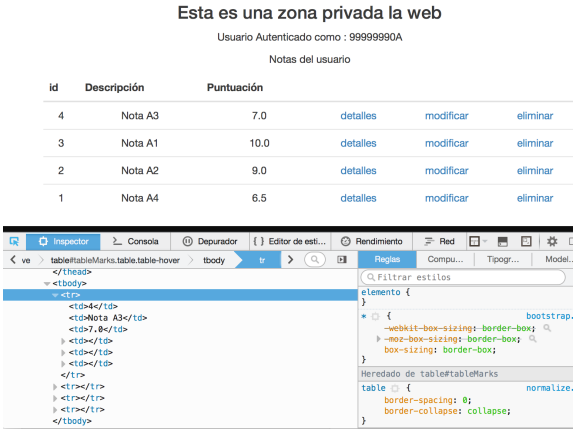
```
//PR12. Loguearse, comprobar que se visualizan 4 filas de notas y desconectarse usando el rol de
estudiante
@Test
@Order(14)
public void PR12() {
  //Vamos al formulario de logueo.
  PO_HomeView.clickOption(driver, "login", "class", "btn btn-primary");
  //Rellenamos el formulario
  PO_LoginView.fillLoginForm(driver, "99999990A", "123456");
  //COmprobamos que entramos en la pagina privada de Alumno
  String checkText =  "Notas del usuario";
  List<WebElement> result = PO_View.checkElementBy(driver, "text", checkText);

  //Contamos el número de filas de notas
  List<WebElement> markList = SeleniumUtils.waitLoadElementsBy(driver, "free", "//tbody/tr",
```

```
PO_View.getTimeout());
   Assertions.assertEquals(4, markList.size());

   //Ahora nos desconectamos  y comprobamos que aparece el menú de registro
   String loginText = PO_HomeView.getP().getString("signup.message", PO_Properties.getSPANISH());
   PO_PrivateView.clickOption(driver, "logout", "text", loginText);
}
```

En este caso una vez identificado el usuario, comprobamos que se vean 4 filas de notas. Para ello empleamos la consulta xpath `"//tbody/tr"` y a continuación comprobaremos que haya 4 objetos WebElement.



2.3.2.4.6.2 PR13: Detalle de una Nota

Para el caso PR13 usaremos el siguiente código:

```
//PR13. Loguearse como estudiante y ver los detalles de la nota con Descripcion = Nota A2.
@Test
@Order(15)
public void PR13() {
   //Comprobamos que entramos en la pagina privada de Alumno
   PO_HomeView.clickOption(driver, "login", "class", "btn btn-primary");
   PO_LoginView.fillLoginForm(driver, "99999990A", "123456");
   String checkText = "Notas del usuario";
   List<WebElement> result = PO_View.checkElementBy(driver, "text", checkText);

   //SeleniumUtils.esperarSegundos(driver, 1);
   //Contamos las notas
   By enlace = By.xpath("//td[contains(text(), 'Nota A2')]/following-sibling::*[2]");
   driver.findElement(enlace).click();
   //Esperamos por la ventana de detalle
   checkText = "Detalles de la nota";
   result = PO_View.checkElementBy(driver, "text", checkText);
   Assertions.assertEquals(checkText,result.get(0).getText());

   //Ahora nos desconectamos comprobamas que aparece el menu de registrarse
   String loginText = PO_HomeView.getP().getString("signup.message", PO_Properties.getSPANISH());
```

```
    PO_PrivateView.clickOption(driver, "logout", "text", loginText);
}
```

El código es bastante claro salvo la consulta xpath necesaria para seleccionar el enlace "Detalle" para la Nota "Nota A2": `"//td[contains(text(), 'Nota A2')]/following-sibling::*[2]`, que hace referencia al segundo "td" después de aquel que contenga el texto "Nota A2" y se mueve a los siguientes dos hermando del elemento seleccionado.

2.3.2.4.6.3 PR14: Agregar una nota

El código para este caso de prueba es el siguiente:

```
//P14. Loguearse como profesor y Agregar Nota A2.
//P14. Esta prueba podría encapsularse mejor ...
@Test
@Order(16)
public void PR14() {
    //Vamos al formulario de login.
    PO_HomeView.clickOption(driver, "login", "class", "btn btn-primary");
    PO_LoginView.fillLoginForm(driver, "99999993D", "123456");
    //Cmmprobamos que entramos en la pagina privada del Profesor
    PO_View.checkElementBy(driver, "text", "99999993D");

    //Pinchamos en la opción de menú de Notas: //li[contains(@id, 'marks-menu')]/a
    List<WebElement> elements = PO_View.checkElementBy(driver, "free", "//li[contains(@id, 'marks-
menu')]/a");
    elements.get(0).click();

    //Esperamos a  que aparezca la opción de añadir nota: //a[contains(@href, 'mark/add')]
    elements = PO_View.checkElementBy(driver, "free", "//a[contains(@href, 'mark/add')]");
    //Pinchamos en agregar Nota.
    elements.get(0).click();

    //Ahora vamos a rellenar la nota. //option[contains(@value, '4')]
    String checkText = "Nota Nueva 1";
    PO_PrivateView.fillFormAddMark(driver, 3, checkText, "8");
    //Esperamos a que se muestren los enlaces de paginación de la lista de notas
    elements = PO_View.checkElementBy(driver, "free", "//a[contains(@class, 'page-link')]");
    //Nos vamos a la última página
    elements.get(3).click();
    //Comprobamos que aparece la nota en la página
    elements = PO_View.checkElementBy(driver, "text", checkText);
    Assertions.assertEquals(checkText, elements.get(0).getText());

    //Ahora nos desconectamos y comprobamos que aparece el menú de registrarse
    String loginText = PO_HomeView.getP().getString("signup.message", PO_Properties.getSPANISH());
    PO_PrivateView.clickOption(driver, "logout", "text", loginText);
}
```

Esta pieza de código muestra la carencia de homogeneidad en el etiquetado de atributos que tiene el código HTML que estamos probando:

- El enlace del menú de notas emplea el atributo id = "marks-menu"
- El enlace de la opción para agregar una nota emplea el atributo href="mark/add".
- Los enlaces de paginación emplean el estilo 'page-link". En este caso obtenemos 3 elementos WebElement y clicamos en el 3: `elements.get(3).click();`

2.3.2.4.6.4 PR15: Eliminar una nota

El código para este caso de prueba es el siguiente:

```
@Test
@Order(17)
public void PR15() {
   //Vamos al formulario de login.
   PO_HomeView.clickOption(driver, "login", "class", "btn btn-primary");
   PO_LoginView.fillLoginForm(driver, "99999993D", "123456");

   //Comprobamos que entramos en la página privada del Profesor
   PO_View.checkElementBy(driver, "text", "99999993D");
   //Pinchamos en la opción de menú de Notas: //li[contains(@id, 'marks-menu')]/a
   List<WebElement>  elements = PO_View.checkElementBy(driver, "free", "//li[contains(@id, 'marks-
menu')]/a");
   elements.get(0).click();
   //Pinchamos en la opción de lista de notas.
   elements = PO_View.checkElementBy(driver, "free", "//a[contains(@href, 'mark/list')]");
   elements.get(0).click();
   //Esperamos a que se muestren los enlaces de paginación la lista de notas
   elements = PO_View.checkElementBy(driver, "free", "//a[contains(@class, 'page-link')]");
   //Nos vamos a la última página
   elements.get(3).click();

   //Esperamos a que aparezca la Nueva nota en la última página
   //Y Pinchamos en el enlace de borrado de la Nota "Nota Nueva 1"
   elements = PO_View.checkElementBy(driver, "free", "//td[contains(text(), 'Nota Nueva 1')]/following-
sibling::*/a[contains(@href, 'mark/delete')]");
   elements.get(0).click();
   //Volvemos a la última página
   elements = PO_View.checkElementBy(driver, "free", "//a[contains(@class, 'page-link')]");
   elements.get(3).click();
   //Y esperamos a que NO aparezca la última "Nueva Nota 1"
   SeleniumUtils.waitTextIsNotPresentOnPage(driver, "Nota Nueva 1",PO_View.getTimeout());

   //Ahora nos desconectamos comprobamos que aparece el menú de registrarse
   String loginText = PO_HomeView.getP().getString("signup.message", PO_Properties.getSPANISH());
   PO_PrivateView.clickOption(driver, "logout", "text", loginText);
}
```

Dado que este caso es un poco más complejo que los anteriores, vamos a indicar el orden de interacción y los métodos reciclados o nuevas consultas xpath empleadas:

1. Vamos a la opción de menú (PO_HomeView.clickOption).
2. Nos logueamos como profesor (PO_LoginView.fillLoginForm).
3. Esperamos a que aparezca la cabecera de datos privados de profesor (PO_View.checkElementBy).
4. Pinchamos en la opción de menú de notas ("//li[contains(@id, 'marks-menu')]/a").
5. Pinchamos en la opción de listado de notas ("//a[contains(@href, 'mark/list')]").
6. Pinchamos en el tercer enlace de paginación ("//a[contains(@class, 'page-link')]").
7. Pinchamos el enlace de borrado de la nota con descripción "Nota Nueva 1" ("//td[contains(text(), 'Nota Nueva 1')]/following-sibling::*/a[contains(@href, 'mark/delete')]").
8. Vamos de nuevo a la última página ya que al borrar una nota la paginación nos lleva a la página 1. ("//a[contains(@class, 'page-link')]).
9. Comprobamos que en esa página no aparezca la descripción "Nota Nueva 1" (SeleniumUtils.waitTextIsNotPresentOnPage).
10. Finalmente nos desconectamos.

Nota: Incluir el siguiente Commit Message ->

"SDI-IDGIT-5.7-Casos Uso VistaPrivada."

OJO: sustituir IDGIT por tu número asignado (p.e. 2223-101):

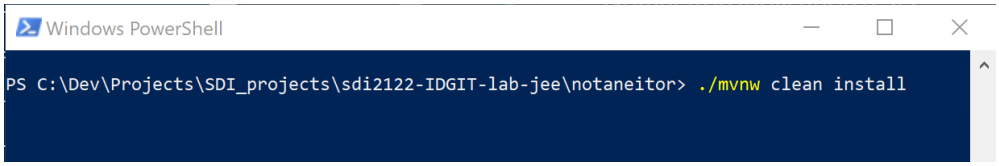
"SDI-2223-101-5.7-Casos Uso VistaPrivada."

(No olvides incluir los guiones y NO incluyas BLANCOS)

2.4 Cuestiones generales

2.4.1 Generación del archivo JAR para despliegue

Para generar el archivo JAR solo hay que ejecutar el siguiente comando por consola el siguiente comando:



Para Mac y otros sistemas operáticos se utilizaría: **mvn clean install**.

No obstante, en algunas ocasiones Maven nos permite generar un archivo FAT Jar para poder ser desplegado de forma autónoma en un servidor remoto. El problema está en

utilizar librerías que dan algún problema si se emplean como dependecias en nuestro proyecto. Por lo tanto, para crear el FAT Jar habría que crear un segundo proyecto donde suprimamos todo lo relativo a pruebas (librerías y código fuente). Y con este segundo proyecto si podemos generar el FAT Jar.

2.4.2 Ejecución de las pruebas varias veces

Dado que las pruebas que hemos implementado en esta práctica alteran el estado de la base de datos, una segunda ejecución conllevaría error en algunas de ellas. Es por ello que o bien inicias la base de datos por código al ejecutar cada prueba o bien reinicias el servidor de base de datos antes de cada prueba. El Jar presentado tiene la propiedad spring.jpa.hibernate.ddl-auto=**create** , por lo que con reiniciar la aplicación serviría.

2.4.3 Refactorización de código

En las pruebas que hemos diseñado e implementado en esta práctica hay código repetido, por lo tanto, para que el proyecto esté lo más limpio posible lo ideal es refactorizarlo y limpiarlo en futuras pruebas.

2.5 Etiquetar proyecto en GitHub

Nota: Etiquetar el proyecto en este punto con la siguiente etiqueta -> **sdi-springboot-p4.**

Para crear una **Release** en un proyecto que está almacenado en un repositorio de control de versiones lo común es utilizar etiquetas (Tags), que nos permitirá marcar el avance de un proyecto en un punto dado (una funcionalidad, una versión del proyecto, etc).

Para crear la etiqueta el proyecto en guíate del ejemplo que realizamos en el guión 2. El nombre de commit será: "***Solución desarrollo de pruebas con Selenium"***

2.6 Resultado esperado en el repositorio de GitHub

Al revisar el repositorio de código en GitHub, los resultados de los commits realizados deberían ser parecidos a estos.

- ⇒ **SDI-2223-101-5.1-ClasePrincipal.**
- ⇒ **SDI-2223-101-5.2-Clases Auxiliares Base.**
- ⇒ **SDI-2223-101-5.3-PO NavView.**
- ⇒ **SDI-2223-101-5.4-Casos Uso VistaHome.**
- ⇒ **SDI-2223-101-5.5-Casos Uso VistaRegister.**
- ⇒ **SDI-2223-101-5.6-Casos Uso VistaLogin.**
- ⇒ **SDI-2223-101-5.7-Casos Uso VistaPrivada.**