Material descargable

egg

WEB AUTOMATION

¿Qué es el Page Object Model?

El **Page Object Model (POM)** es un **patrón de diseño** utilizado en la automatización de pruebas que te permite estructurar tu código de manera organizada y mantenible. En lugar de mezclar la lógica de las pruebas con la interacción de la aplicación web, POM sugiere separar estas responsabilidades mediante la creación de **objetos de página (Page Objects)**.

Cada **Page Object** representa una página específica de la aplicación y encapsula:

- **Los localizadores de los elementos web** (botones, campos de texto, enlaces, etc.).
- **Los métodos que interactúan con estos elementos** (como ingresar texto o hacer clic en un botón).

Beneficios de utilizar Page Object Model

Implementar POM en tus pruebas te proporciona ventajas clave:

- **Mantenibilidad:** Si un elemento de la página cambia, solo necesitas actualizar su localizador en un solo lugar.
- **Legibilidad:** Los tests son más fáciles de entender, ya que reflejan el flujo lógico de la aplicación sin exponer detalles técnicos innecesarios.
- **Reutilización:** Puedes emplear los métodos de los Page Objects en múltiples pruebas, evitando duplicación de código.

Pasos para definir un Page Model

Para optimizar tus pruebas con POM, sigue estos pasos:

1. **Identifica las páginas de la aplicación.**
2. **Crea una clase de Page Object** para cada página, definiendo sus localizadores y métodos de interacción.
3. **Implementa métodos** que realicen acciones en la página (por ejemplo, completar formularios o navegar a otra sección).
4. **Escribe tus scripts de prueba** utilizando los Page Objects en lugar de interactuar directamente con los elementos de la interfaz.

¿Qué es el Page Factory?

El **Page Factory** es una extensión del Page Object Model en Selenium que mejora la forma en que gestionas los elementos web dentro de tus pruebas. Su principal ventaja es la inicialización automática de los elementos, lo que simplifica el código y mejora la eficiencia de ejecución.

Cuando usas **Page Factory**, empleas la anotación `@FindBy` para definir los localizadores de los elementos web. Luego, al instanciar la página en tu prueba, **Selenium los encuentra y los inicializa automáticamente**, evitando que debas buscarlos manualmente en cada prueba.

Beneficios de Page Factory

- **Código más limpio y estructurado.**
- **Facilidad de mantenimiento** cuando la interfaz cambia.
- **Menos código repetitivo**, ya que la inicialización de elementos es automática.

Implementación de Page Factory en Selenium

1. **Configuración del Driver:** Define una clase que administre el driver del navegador:

```
public class Driver {
    private WebDriver driver;
    public Driver(WebDriver driver, String url) {
       this.driver = driver;
       PageFactory.initElements(this.driver, this);
       this.driver.manage().window().maximize();
       this.driver.get(url);
    }
    public String getTitle() {
       return this.driver.getTitle();
    }
 }
```

2. **Creación de la clase de Page Object:** Extiende la clase `Driver` y define los elementos que deseas automatizar con `@FindBy`:

```
public class HomePage extends Driver {
    public HomePage(WebDriver driver, String url) {
       super(driver, url);
    }

    @FindBy(id = "searchField")
    private WebElement searchField;
```

```
    @FindBy(css = "button#search")
    private WebElement searchButton;

    public void search(String text) {
       searchField.sendKeys(text);
       searchButton.click();
    }
 }
```

3. **Definición de pruebas con Page Factory:** Configura la ejecución de las pruebas asegurando la correcta inicialización y cierre del navegador:

```
private HomePage home;
private WebDriver driver;

@BeforeEach
public void setUp() {
   System.setProperty("webdriver.chrome.driver", "/ruta");
   driver = new ChromeDriver();
   home = new HomePage(driver, "http://www.google.com");
}

@AfterEach
public void tearDown() {
   driver.quit();
}

@Test
public void testingSearch() {
   home.search("automation");

   WebDriverWait waitSearch = new WebDriverWait(driver, Duration.ofSeconds(10));
   waitSearch.until(ExpectedConditions.titleContains("automation"));

   Assertions.assertTrue(driver.findElements(By.cssSelector("div#search
div.g")).size() > 0);
}
```

El uso de **Page Factory** en el **Page Object Model** optimiza la automatización de pruebas, asegurando un código más limpio, mantenible y reutilizable. A medida que te familiarices con este enfoque, notarás cómo mejora la estructura de tus pruebas y facilita la gestión de cambios en la interfaz de usuario.

Para profundizar en **Page Object Model (POM)**, puedes dirigirte a la documentación oficial de Selenium, donde encontrarás información detallada y ejemplos sobre la implementación de estos patrones en la automatización de pruebas. **Selenium Official Documentation - Page Object Model**