Testing I
Profesor: Daniel Tejerina

Módulo 7: Introducción a Automation

Sesión 22: Fundamentos de automatización de la prueba
Diciembre 7 de 2021

Pruebas automatizadas
Es el proceso de ejecutar varias pruebas una y otra vez sin intervención humana utilizando una herramienta de automatización.

Pruebas que se pueden automatizar:
- Pruebas unitarias
- Pruebas de API
- Pruebas de Interfaz gráfica

Otras:
- Pruebas de Regresión
- Pruebas de Integración
- Pruebas de seguridad
- Pruebas de compatibilidad en diferentes navegadores

Patrones de diseño
son soluciones probadas y documentadas a problemas comunes de desarrollo de software.

Patrones de diseño en automatización

Screenplay
- Tiene un enfoque de Behavior Driven Development (BDD), estrategia de desarrollo que se enfoca en prevenir defectos en lugar de encontrarlos en un ambiente controlado.
- Presenta un alto desacoplamiento de la interfaz de usuario.
- Propone trabajar en términos de tareas y no de interfaz de usuario.
- Cumple con los principios SOLID.

Page Object
- Es un patrón de diseño que representa los componentes web o página web en una clase.
- Se utiliza en las pruebas automatizadas para evitar código duplicado y mejorar el rendimiento de las mismas.
- No cumple con los principios SOLID.

Principios SOLID

Acrónimo introducido por Robert C. Martin para definir los 5 principios de la POO:
- Single responsibility
- Open-closed
- Liskov substitution
- Interface segregation
- Dependency inversion

Los principios SOLID son guías, buenas prácticas, que pueden ser aplicadas en el desarrollo de software para ayudar a escribir un mejor código: más limpio, mantenible y escalable.

Page Object Model (POM)
Es la implementación del patrón de diseño Page Object utilizado en la automatización de pruebas. Se basa en la idea de representar cada una de las pantallas que componen al sitio web o la aplicación que interesa probar como una serie de objetos que encapsulan las características (localizadores) y funcionalidades representadas en la página.

Las páginas web se representan como clases y los elementos de la página se definen como variables en la clase. Todas las interacciones de usuario se pueden implementar como métodos en la clase. Estas interacciones generalmente se dividen en acciones y validaciones. Las acciones reflejan los pasos que el usuario debe realizar dentro del sitio web o aplicación bajo prueba. Las validaciones son métodos que reflejan la confirmación de que el sistema se comporta como se espera ante una acción específica.

¿Qué es Selenium?
Es un framework destinado a la automatización web que consiste en desarrollar scripts que, mediante algún lenguaje de codificación determinado, permite ejecutar un flujo de navegación fijo.

Selenium IDE
Selenium IDE o Selenium Recorder es una herramienta de automatización que permite grabar, editar y depurar pruebas, sin necesidad de usar un lenguaje de programación.

Selenium WebDriver
Es una herramienta que permite automatizar pruebas UI de aplicaciones web. Permite simular la manera en que los usuarios interactúan con la aplicación. Soporta Java, C#, Python, Ruby, PHP, JavaScript, entre otros.

Selenium Grid
Permite diseñar pruebas automatizadas para aplicaciones web en diversas plataformas y posibilita la ejecución de pruebas en diversos servidores en paralelo.

Selenium IDE
Es un entorno de pruebas de software para aplicaciones basadas en la web que permite realizar tareas de *Record&Play* de flujos de pruebas. Los flujos grabados quedan contenidos en un script que se puede editar y parametrizar para adaptarse a los diferentes casos, y su ejecución se puede repetir tantas veces como se quiera.
Permite referenciar objetos del DOM en base al ID, CSS, nombre o a través de XPath.

Instalar Selenium IDE
(Se instala como extensión de Chrome):
https://www.selenium.dev/selenium-ide

Para grabar un test:
- Click en "Record a new test in a new project"
- Especificar el nombre del proyecto
- Especificar una url base y click en "Start Recording"
- Al finalizar dar click en el botón Stope Selenium
- Ingresar el nombre del caso de prueba

Sesión 23. Automatización de la prueba
Diciembre 9 de 2021.

Selenium WebDriver

- Crear la carpeta del proyecto
- Crear un archivo test.js
- Abrir VSC
- Instalar Selenium

npm install selenium-webdriver

(Se genera el archivo package-lock.json)
- Descargar el archivo binario de Chrome driver (?).
- Para ello se requiere verificar la versión de Chrome
- Descargar la versión correspondiente según la versión de Chrome y el SO: https://chromedriver.storage.googleapis.com/index.html

- Descomprimir el archivo descargado
- Mover el archivo .exe a la carpeta del proyecto. Esto solo permite que se ejecute selenium en el proyecto.
- Alternativamente se puede alojar en alguna carpeta de Windows incluida en el Path para que esté disponible para todos los poryectos.

```
// Este test permite verificar que el login en facebook es exitoso
const { Builder, Capabilities } = require("selenium-webdriver");
// otra versión: const { Builder, Key, By, Capabilities } = require("selenium-webdriver");
const chromeCapabilities = Capabilities.chrome();
const assert = requiere('assert')

async function TC_001 () {

        chromeCapabilities.set('chromeOptions', { args: ['--headless']});
        let driver = await new
Builder().forBrowser('chrome').withCapabilities(chromeCapabilities).build();

        await driver.get('https://www.facebook.com/');
        await driver.manage().window().maximize(); // Maximiza la ventana

        await driver.sleep(10000); // Genera una pausa en la ejecución para permitir visualizarla

        await driver.findElement(By.id('email')).sendKeys('mao_pineda@hotmail.com');
        await driver.findElement(By.id('pass')).sendKeys('ennvapdscucdm');
        await driver.findElementt(By.xpath("//button[@type='submit']")).click(); // No se
especifica el botón por su id (podría ser dinámico), sino por su propiedad xpath

        await driver.sleep(2000);
```

```
        let buttonTextCompare = await
driver.findElement(By.id('checkpointSubmitButton')).getAttribute('value');
        assert.strictEqual(buttonTextCompare, "Continuar"); // Verifica que en la nueva pantalla
el botón tiene el texto 'Continuar'
        console.log("Login successfully");

        await driver.sleep(10000);
        awair driver.quit(); // Cierra el navegador
}

TC_001();
```

- Ejecutar el archivo:
node test.js

Sincrónico

Selenium WebDriver. Instalación y pasos iniciales
https://www.npmjs.com/package/selenium-webdriver

Ejemplo youtube:

```
const { Builder, Capabilities } = require("selenium-webdriver");
const chromeCapabilities = Capabilities.chrome();

const assert = require('assert');

async function youtube_dh(){
  chromeCapabilities.set('chromeOptions', {arg:['--headless']});
  let driver = await new
Builder().forBrowser('chrome').withCapabilities(chromeCapabilities).build();

  await driver.get('https://www.youtube.com')

}
youtube_dh();
```

Driver:
http://chromedriver.storage.googleapis.com/index.html?path=96.0.4664.35/

Sesión 25. Test Plan
Diciembre 14 de 2021.

Plan de Pruebas

Un plan de prueba es un documento detallado que describe la estrategia de prueba, los objetivos, los recursos necesarios, el cronograma y los criterios de éxito para probar una nueva característica o pieza de software específica.

¿Qué se debe incluir en un Plan de Prueba?

1. Cobertura

- Introducción
- Objetivos
- Alcance de alto nivel
- Cronograma
- ¿Qué pruebas se van arealizar? ¿Por qué?

2. Métodos

Explicar la estrategia de prueba.
- ¿Qué reglas seguirán las pruebas?
- ¿Qué métricas se van a recopilar y a qué nivel?
- ¿Cuántas configuraciones o entornos se van a probar?
- ¿Existen requisitos o procedimientos especiales que deba probar?

¿Cuáles son los criterios de aprobación/reprobación para cada prueba?
- Criterio de salida
- Criterios de suspensión
- Requisitos de reanudación

Enumerar suposiciones y riesgos.
- ¿qué supone que va a suceder y cuáles son algunos de los riesgos que se enfrentarán durante la prueba?

Describir recursos y cronograma del proyecto
- ¿Quién está a cargo de las pruebas y qué recursos necesitan (tanto técnicos como humanos)?
- ¿Cuándo se realizarán las pruebas y durante cuánto tiempo?

3. Responsabilidades

- ¿Cuáles son los entregables de prueba requeridos?

5 Pasos para crear un Test Plan

- Analizar el producto o función
- Diseñar las estrategias de prueba
- Definir objetivos y criterios
- Planificar el entorno de prueba

- Ejecutar y realizar seguimiento

Contenido del Plan de Test (Template)

1. Introducción
1.1. Alcance
1.2. Referencias
1.3. Definiciones y abreviaturas

2. Objetivos del Test
2.1. Antecedentes
2.2. Objetivo de la Evaluación
2.3. Motivaciones del Test

3. Items a testear

4. Métodos

5. Casos de pruebas
a. Estrategia
b. Casos

6. Criterios de aceptación
6.1. Plan de Test
6.1.1. Criterio de entrada del Plan de Test
6.1.2. Criterio de salida del Plan de Test
6.1.3. Criterio de suspensión y reanudación
6.2. Ciclo de Test

7. Entregables
7.1. Reporte de cubrimiento del Testing
7.2. Log de incidentes y Pedidos de cambio
7.3. Scripts de Test
7.4. Entregables adicionales

8. Recursos requeridos

9. Responsabilidades

10. Planificación temporal de áreas

Sincrónico

Selenium. Encontrar elementos web:
https://www.selenium.dev/documentation/webdriver/elements/finders/

Sesión 26. Full-Stack Tester
Diciembre 16 de 2021.

Frameworks de automatización de pruebas

Un framework define un conjunto de reglas/pautas o mejores prácticas que podemos seguir de manera sistemática para lograr los resultados deseados.

Tipos de frameworks para la automatización de pruebas

- Framework de secuencias de comendos lineal

Conocido como "grabación y reproducción". En este framework el tester registra manualmente cada paso (navegación y entradas del usuario) e inserta puntos de control (pasos de validación) en la primera iteración. Luego, reproduce el script grabado en las iteraciones siguientes.
Ejemplos: Selenium GRID, UFT, Test Complete, Sahi.

- Framework de prueba modular

Se divide la aplicación en varios módulos y se crean scripts de prueba individualmente. Estos se pueden combinar para hacer scripts de prueba más grandes utilizando uno maestro para lograr los escenarios requeridos. El script maestro se utiliza para invocar los módulos individuales para ejecutar escenarios de prueba de un extremo a otro (E2E).

- Framework de prueba basado en datos

La lógica del caso de prueba reside en los scripts de prueba, pero los datos de prueba se separan y se mantienen fuera de los scripts de prueba. Los datos de prueba se leen de los archivos externos (archivos de Excel, archivos de texto, archivos CSV, fuentes ODBC, objetos DAO, objetos ADO) y se cargan en las variables dentro del script de prueba. Las variables se utilizan tanto para los valores de entrada como para los valores de verificación. Los scripts de prueba se preparan utilizando Linear Scripting o Test Library Framework.
Ejemplo: UFT, Specflow (C#)

- Framework de prueba basado en palabras clave

Conocido como prueba basada en tablas o prueba basada en palabras de acción. El desarrollo de este framework requiere tablas de datos y palabras clave. La funcionalidad de la aplicación bajo prueba se documenta en una tabla, así como en instrucciones paso a paso para cada prueba.
Ejemplo: Robot Framework (Python).

- Framework de prueba híbrido

Es la combinación de dos o más frameworks mencionados anteriormente. La industria generalmente utiliza el framework de palabras clave en una combinación con el de descomposición de funciones.
Ejemplo: Karate-DSL, Citrus, etc.

- Framework de prueba de desarrollo guiado por el comportamiento
Su propósito es crear una plataforma que permita a todos (analistas de negocios, desarrolladores, probadores, etc.) participar activamente. Requiere una mayor colaboración entre los equipos de desarrollo y prueba, pero no requiere que los usuarios estén familiarizados con un lenguaje de programación. se utiliza un lenguaje natural no técnico para crear especificaciones de prueba (Gherkin).
Ej.: JBehave (Java) , Cucumber , Specflow (C#), Serenity, etc.

Sesión 27. Fin.
Diciembre 17 de 2021

QA manual
Conoce el proceso fundamental de testing (ciclo de vida de las pruebas de software - SDLC) y las técnicas para crear casos de prueba. Su principal tarea es la de crear y ejecutar los casos de prueba. A través de las pruebas identifica, reporta y realiza el seguimiento de los defectos encontrados.

QA especialista
Se enfoca en la integración continua y la entrega continua. Posee el dominio de herramientas de soporte a actividades de prueba y conoce el proceso para realizar pruebas end to end.

QA lead
Se enfoca en la entrega de valor al cliente y domina el proceso de testing. Cuenta con habilidades blandas que lo ayudan a organizar el equipo de trabajo, aplicando conceptos y técnicas ágiles para mejorar los procesos de prueba y asegurándose de llevar adelante buenas prácticas.

QA automation
Posee los conocimientos para implementar herramientas de prueba para automatizar casos de prueba funcionales y no funcionales. A su vez, puede identificar cuándo y por qué elegir una herramienta y caso de prueba para el proceso de automatización. Por otro lado, conoce los distintos frameworks de desarrollo y cómo implementarlos. Además, implementa y realiza el seguimiento de buenas prácticas durante el proceso de automatización.