Universidad Tecnológica de Tehuacán

Trabajo:
Herramientas de prueba
Curso:
Evaluación y mejora para el desarrollo de software

Integrantes:

Maria Fernanda Mendoza Ayala

Emilio Giovanny Rivera Falcon

Estefania Martinez Cadena

Daniel Sandoval Olmos

Juan Heber Rojas

https://drive.google.com/file/d/1U1gQSVaSitlHQ9EeNABB7dalXJB-9lZe/view?usp=drive_link

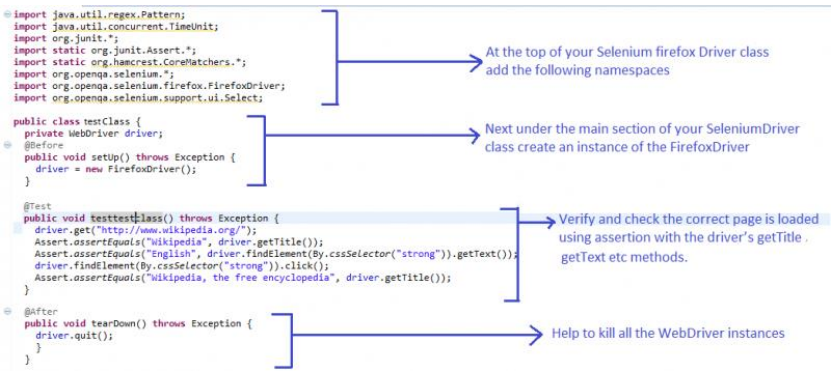
Fecha :

23/11/2023

Contenido

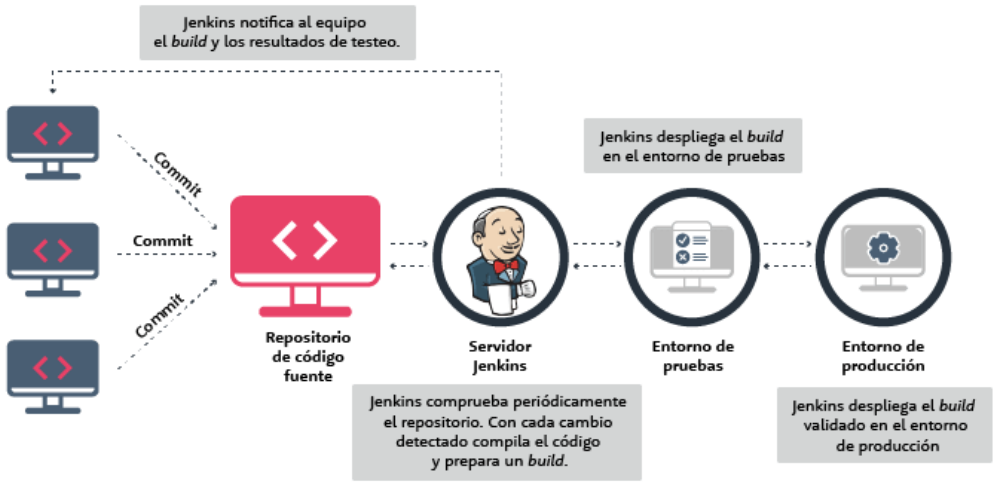
Herramientas de sistema abierto que permiten ejecutar y simular los casos de prueba en sistemas Web

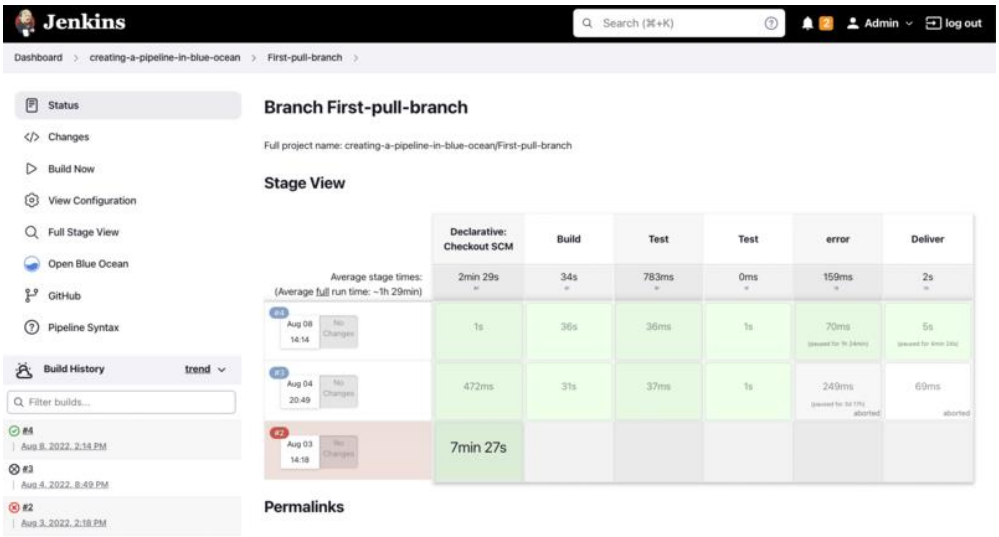
Selenium: Para pruebas de interfaz de usuario en aplicaciones web.
RestAssured: Para pruebas de API REST en Java.
Postman: Herramienta de desarrollo de API con funciones de prueba.



Jenkins: Para integración continua y ejecución automatizada de pruebas.
Travis CI: Servicio de integración continua para repositorios de GitHub.

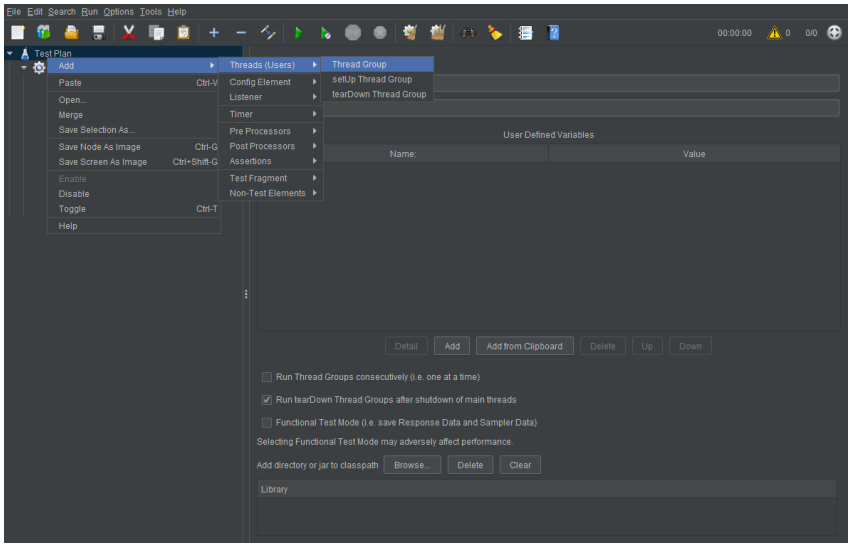
GitLab CI: Herramienta de integración continua integrada con GitLab.





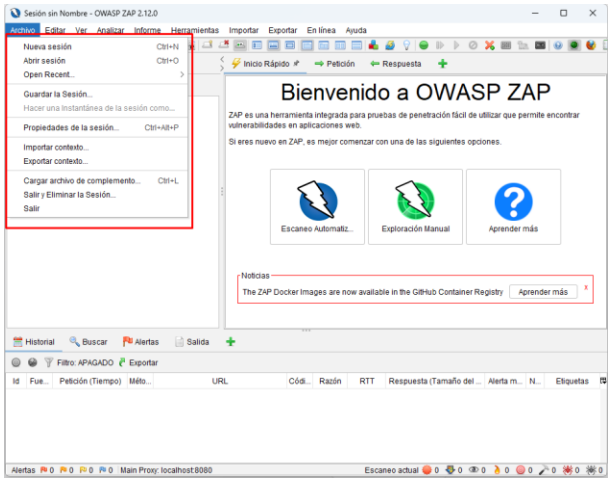
Apache JMeter: Para pruebas de carga y rendimiento.

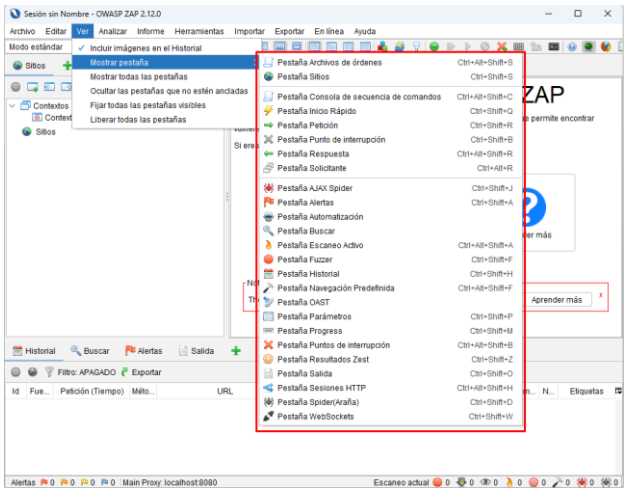
Locust: Marco de pruebas de carga en Python.
Gatling: Herramienta de pruebas de rendimiento basada en Scala y Akka.
Pruebas de seguridad.



OWASP ZAP: Herramienta de prueba de seguridad para aplicaciones web.

Burp Suite: Plataforma integral para pruebas de seguridad de aplicaciones web.
Nessus: Escáner de vulnerabilidades para redes y aplicaciones.
Pruebas de regresión.

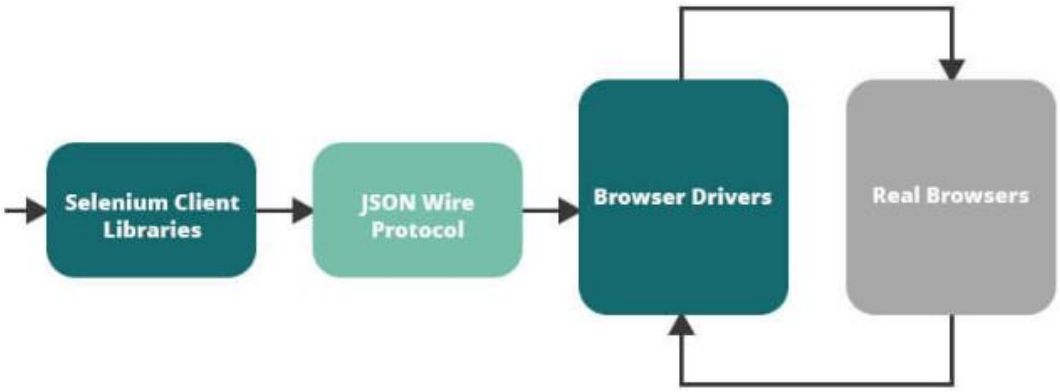




Selenium WebDriver: Para pruebas automatizadas de interfaces de usuario web.

Cypress: Herramienta de prueba de extremo a extremo para aplicaciones web modernas.

JUnit/TestNG con Selenium: Para pruebas de regresión en Java.
Pruebas de carga y estré.



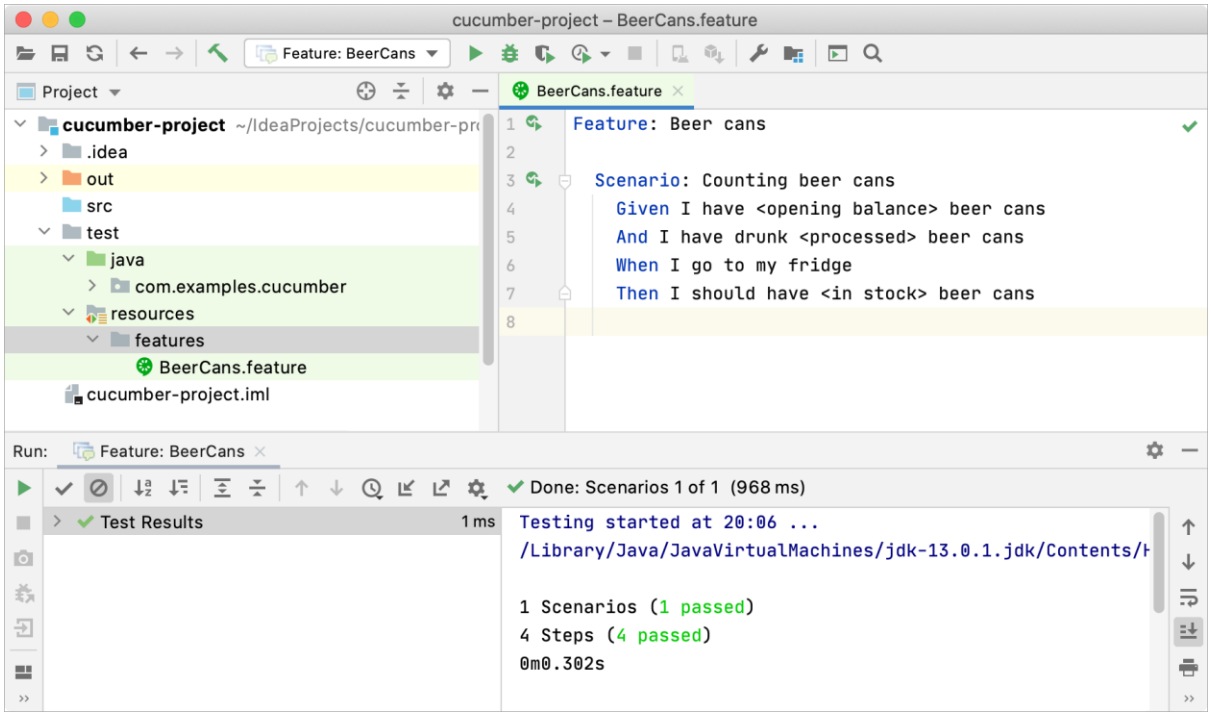
:

Apache JMeter: También se usa para pruebas de carga.

LoadRunner: Herramienta de Micro Focus para pruebas de carga y rendimiento.

BlazeMeter: Plataforma en la nube para pruebas de rendimiento y carga.
Pruebas de aceptación del usuario (UAT):

Cucumber: Herramienta de automatización de pruebas basada en comportamientos.
Robot Framework: Marco de automatización de pruebas de código abierto.



Interpretación de los resultados

Realizamos pruebas de software a nuestra página web del restaurante Pollos Mareados y los resultados fueron los siguientes.

Los tiempos de respuesta en todos los navegadores fueron óptimos y con una respuesta acorde a lo esperado. Él alojamiento del sitio web en el servidor tuvo algunos fallos, pero fueron corregidos y la página siguió funcionando. Otro error encontrado fue una falla a la hora de inicio de sesión, la causa de este problema fue que el servidor en el cual se encontraba alojado la página web se cayó, lo que provocó el fallo, para solucionarlo el equipo se puso en contacto con el proveedor de alojamiento para la actualización del pago.

La página web fue completamente responsiva en todos los dispositivos en donde se probó y no se encontraron problemas de visualización.

Se identificaron problemas relacionados con la carga de imágenes, lo que provocaba una falta de apoyo visual en el menú del sitio, dificultando así la experiencia del usuario, si bien las imágenes tenían el tamaño y posición adecuada, no logran visualizarse o cargar correctamente. Esos errores fueron identificados en el código debido a un problema con las rutas establecidas, la ruta se había escrito mal, lo que provocaba que la imagen no se encontrará cargada, al reescribirse la ruta, la imagen no tuvo ningún problema en aparecer en la página web, se le dio seguimiento al problema hasta solucionarlo y las imágenes y fotos de los productos cargarán de manera óptima y como era requerido.

El equipo de testing verifica los errores y determina que los problemas se deben a errores en el código de la página web. El equipo de testing puede corregir los errores y volver a probar la página web.

Los errores encontrados no eran tan graves, aun así, afectan al funcionamiento de la página web.

A pesar de ello los errores eran fáciles de identificar y corregir. La corrección de los errores no requirió cambios importantes en el código de la página web.

Identificar las herramientas de acceso abierto que permiten ejecutar y simular los casos de prueba en sistemas móviles

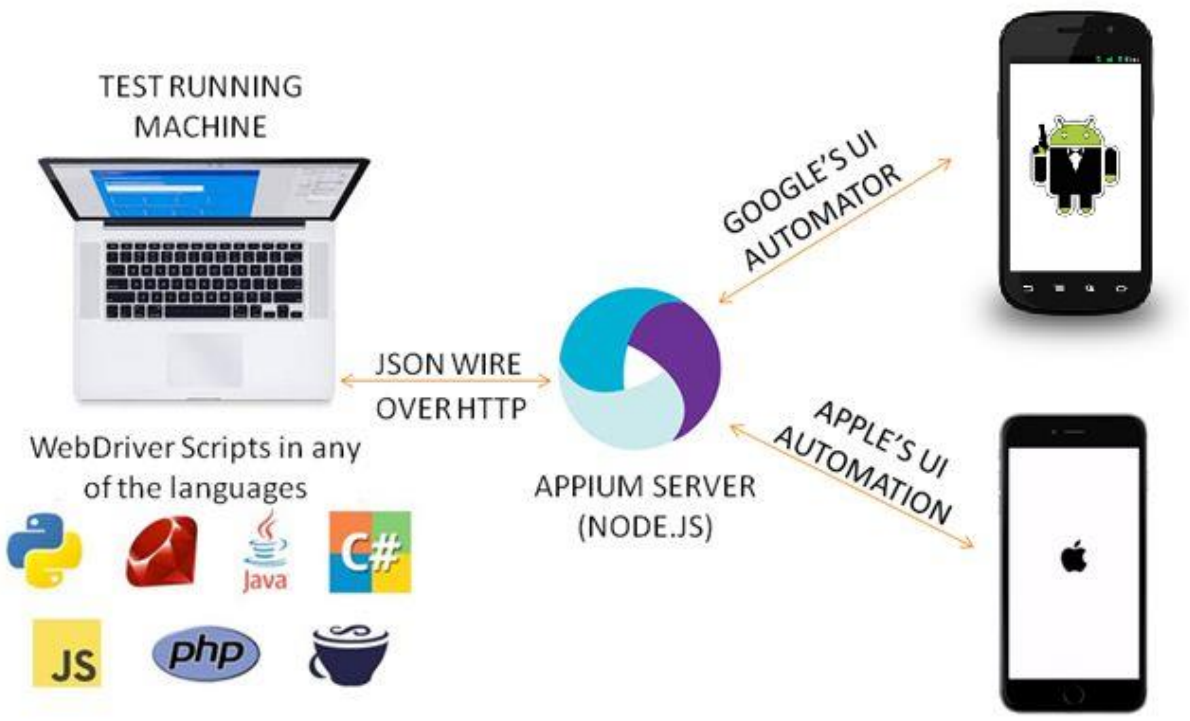
Appium:

Appium es una herramienta de automatización de pruebas de código abierto que se utiliza para probar aplicaciones móviles en plataformas como Android e iOS.

Appium es el estándar en automatización de pruebas para móviles. Se trata de un framework open source que permite probar aplicaciones nativas, híbridas o, como en nuestro caso, aplicaciones web. Se dice de Appium que es como selenium, pero para aplicaciones móviles.

Características:

- Soporta múltiples lenguajes de programación como Java, Python, C#, etc.
- Permite probar aplicaciones nativas, híbridas y basadas en la web.
- Es compatible con emuladores y dispositivos reales.**}**



Instalar Appium

Instalar Appium es tan fácil como ejecutar un único comando NPM:

```
npm i --location=global appium
```

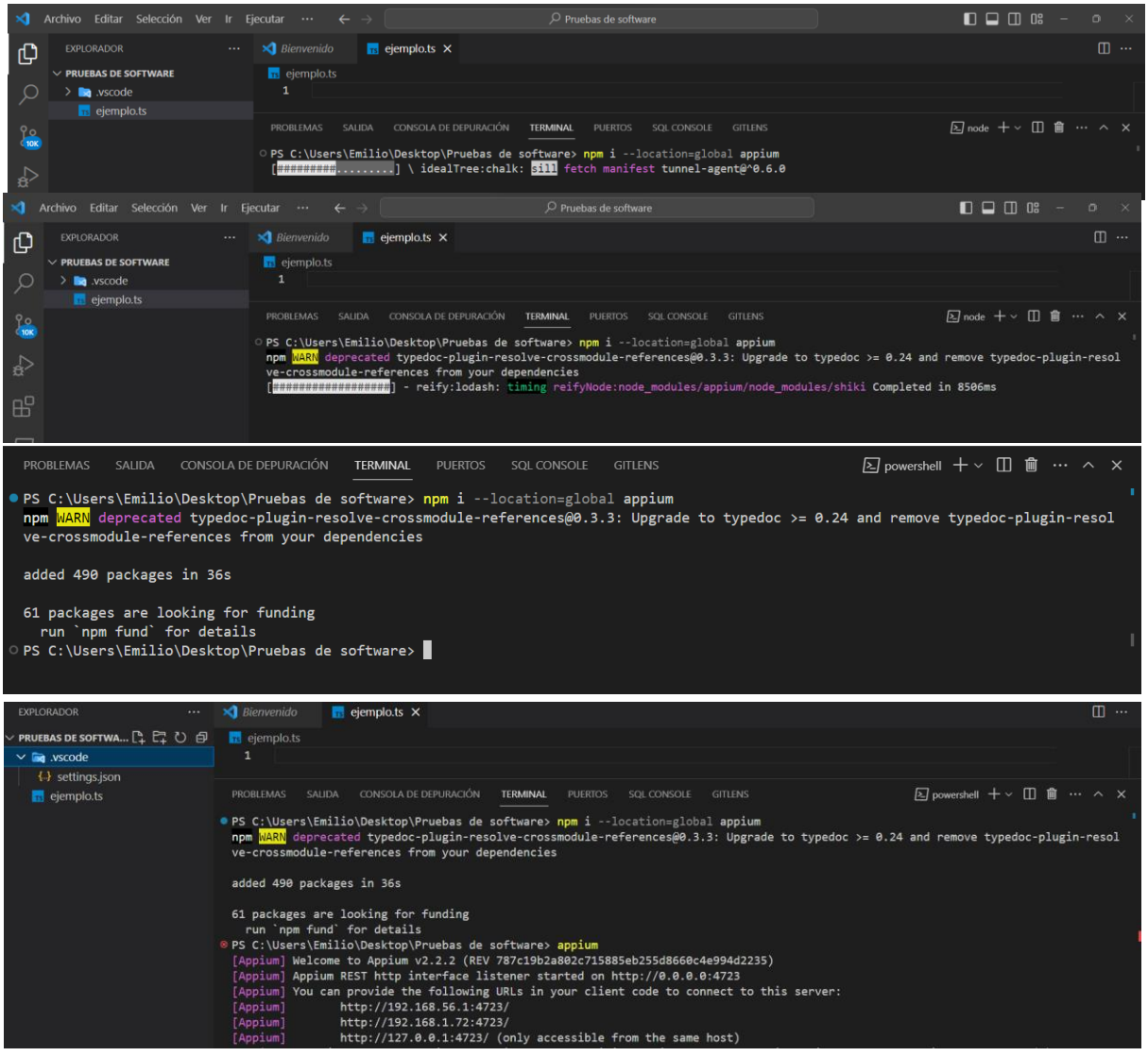
Este comando instala Appium globalmente en su sistema para que pueda acceder a él desde la línea de comando simplemente ejecutando el `appium` comando. Continúe y ejecútelo ahora:

```
appium
```

Deberías ver algún resultado que comience con una línea como esta:

```
[Appium] Welcome to Appium v2.0.0
```

¡Eso es todo! Si obtiene este tipo de resultado, el servidor Appium está en funcionamiento.



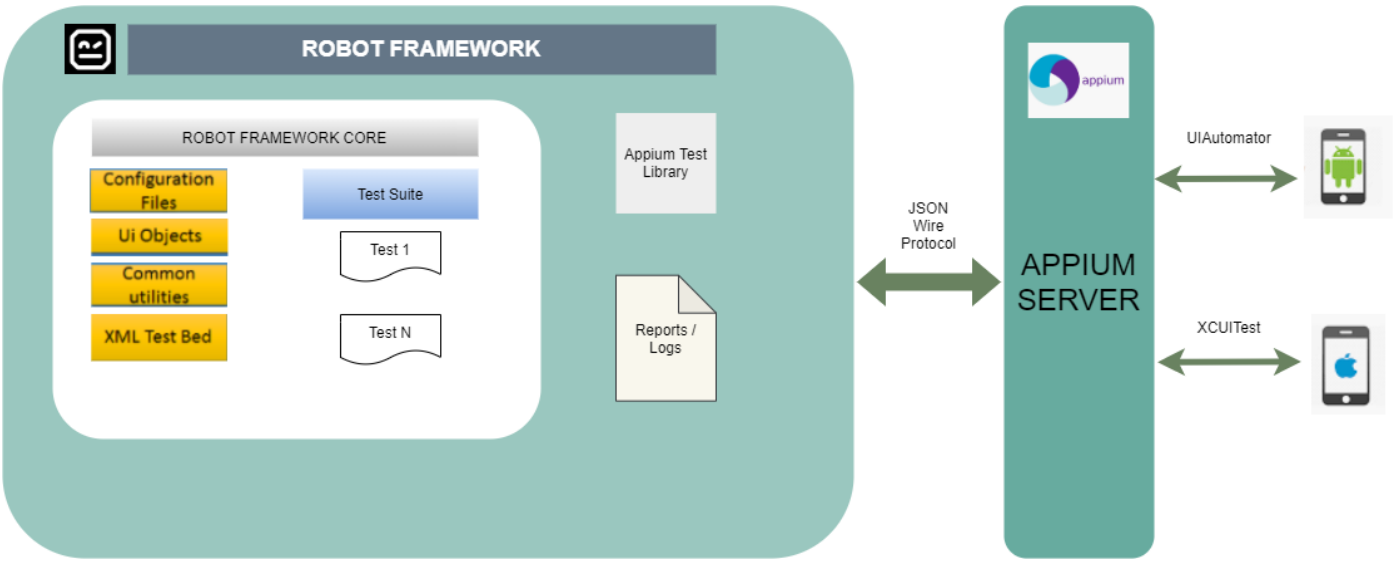
Robot Framework:

Robot Framework es un marco de automatización de pruebas genérico y de código abierto que admite la automatización de pruebas de aplicaciones móviles.

Es un entorno de trabajo de automatización de prueba genérico para testaje de aceptación y desarrollo de prueba de aceptación. Es un marco de trabajo basado en palabras clave que usa una sintaxis de datos de prueba tabular.

Características:

- Puede utilizarse con bibliotecas específicas para pruebas móviles como Appium.
- Permite escribir casos de prueba en un formato fácil de entender llamado "tabla de datos".



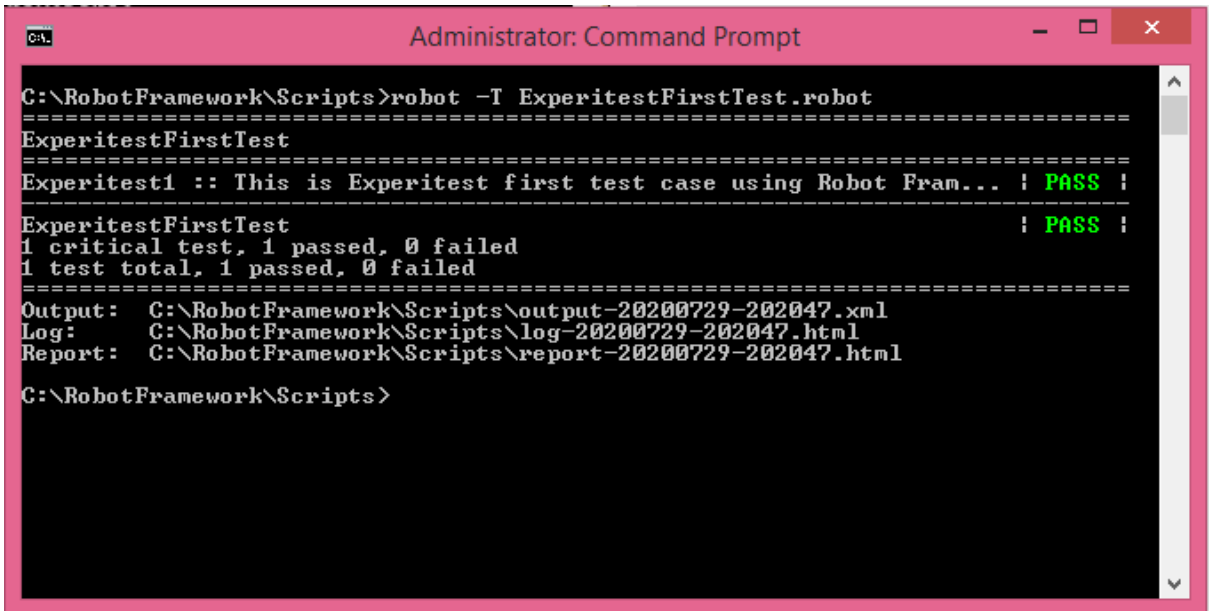
Robot Framework se implementa con Python, por lo que es necesario tener Python instalado.
En máquinas con Windows, asegúrese de agregar Python a PATH durante la instalación.

Instalar Robot Framework con pip es simple:

```
pip instalar robotframework
```

Para comprobar que la instalación fue exitosa, ejecute

```
robot --versión
```


Administrator: Command Prompt
C:\RobotFramework\Scripts>robot -T ExperitestFirstTest.robot
==
ExperitestFirstTest
==
Experitest1 :: This is Experitest first test case using Robot Fram... | PASS |

ExperitestFirstTest | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==
Output: C:\RobotFramework\Scripts\output-20200729-202047.xml
Log: C:\RobotFramework\Scripts\log-20200729-202047.html
Report: C:\RobotFramework\Scripts\report-20200729-202047.html
C:\RobotFramework\Scripts>

DETOX

Detox, desarrollado por Wix, es un marco de prueba móvil diseñado para pruebas de un extremo a otro de aplicaciones móviles creadas con React Native en mente, específicamente en plataformas iOS y Android. Le permite simular las interacciones del usuario (UI) con una aplicación móvil y verificar los comportamientos correctos y los requisitos funcionales de una sola vez.

INSTALACIÓN

El setup completo de Detox se describe en este link:

https://github.com/wix/Detox/blob/master/docs/Introduction.GettingStarted.md

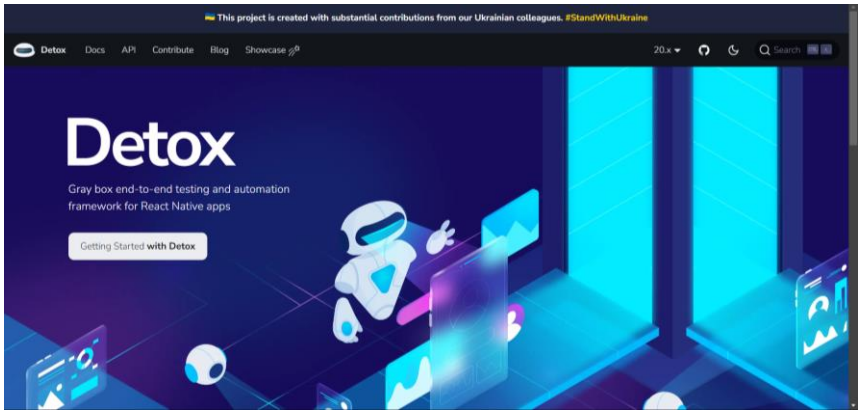
Los pasos principales que debe seguir un desarrollador para usar Detox, con el fin de crear y ejecutar las descripciones de las pruebas. También se puede configurar para ser utilizado como parte del pipeline de CI.

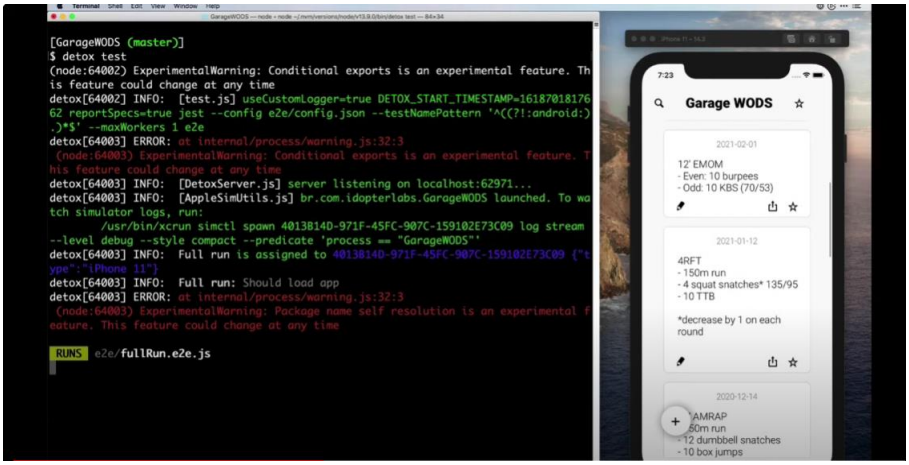
Detox CLI debe instalarse globalmente:

```
# Install detox globally
npm install -g detox-cli
```

Detox puede verse como un marco de prueba de caja gris porque le permite probar el comportamiento y el estado internos de una aplicación además de las interacciones de la interfaz de usuario (UI).

Una de sus características únicas es su capacidad de sincronizarse con la aplicación bajo prueba. Esto significa que el marco espera a que la aplicación quede inactiva antes de continuar con el siguiente paso de prueba, lo que puede ayudar a evitar problemas de sincronización y resultados de prueba incorrectos





ESPRESSO

Espresso es un framework de testing open source lanzado por Google el cual provee una API que permite crear pruebas de Interfaz de usuario (de ahora en adelante UI por sus siglas en inglés) para simular interacciones de usuarios en una aplicación Android

Usa Espresso para escribir pruebas de la IU de Android concisas, eficaces y confiables.

En el siguiente fragmento de código, se muestra un ejemplo de una prueba de Espresso:

```
@Test
public void greeterSaysHello() {
    onView(withId(R.id.name_field)).perform(typeText("Steve"));
    onView(withId(R.id.greet_button)).perform(click());
    onView(withText("Hello Steve!")).check(matches(isDisplayed()));
}
```

INSTALACIÓN E INTERACCIÓN

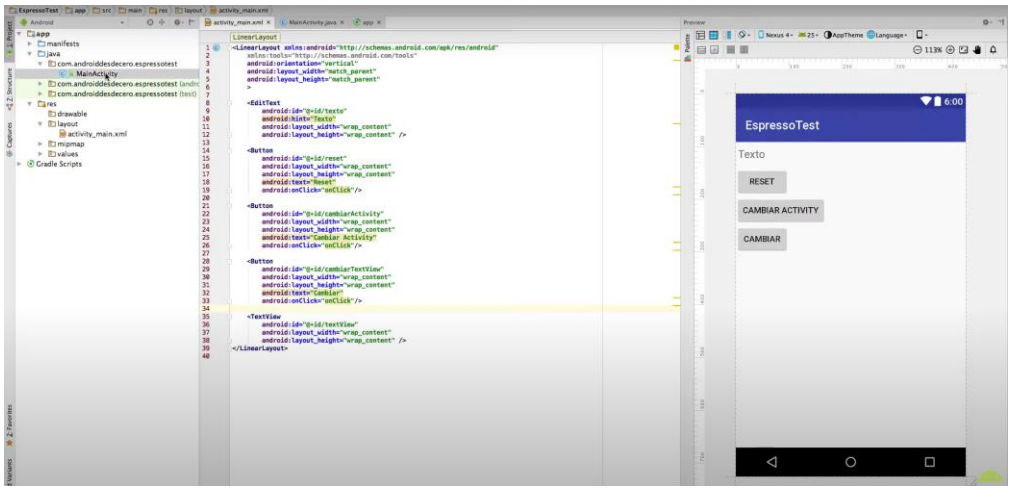
Para el caso de este tutorial, usted ejecutará pruebas sobre una aplicación de ejemplo llamada Habitica. Esta aplicación permite hacer un seguimiento a tareas diarias y hábitos haciendo uso de gamificación. Es un proyecto de código abierto desarrollado por HabitRPG el cual cuenta con más de un millón de descargas desde Google Play Store.

La API principal es pequeña, predecible y fácil de aprender, pero puedes personalizarla. Las pruebas de Espresso exponen claramente las expectativas, las interacciones y las afirmaciones sin la distracción del contenido estándar, la infraestructura personalizada o los complicados detalles de implementación que se interponen en el camino.

Las pruebas de Espresso se ejecutan con una rapidez óptima. Te permiten dejar atrás esperas, sincronizaciones, tiempos inactivos y sondeos, mientras manipulan y afirman en la IU cuándo están en reposo.

Público objetivo
Espresso está dirigido a desarrolladores, que creen que las pruebas automatizadas son una parte integral del ciclo de vida del desarrollo. Si bien Espresso se puede usar para pruebas de caja negra, quienes están familiarizados con la base de código bajo en modo de prueba pueden aprovecharlo al máximo.

Ejemplo de cómo se visualiza un proyecto de Espresso



junit:

JUnit es un framework de pruebas unitarias de código abierto para la plataforma Java. Es una de las herramientas de pruebas unitarias más populares y utilizadas en el mundo.

JUnit proporciona una API sencilla y fácil de usar que permite a los desarrolladores escribir pruebas unitarias para sus clases y métodos. Las pruebas unitarias se utilizan para verificar el comportamiento de una unidad de código, como una clase o un método.

JUnit proporciona una serie de características que facilitan la escritura de pruebas unitarias, incluyendo:

- Afirmaciones: JUnit proporciona una serie de afirmaciones que permiten a los desarrolladores verificar el resultado de una operación.
- Anotaciones: JUnit utiliza anotaciones para marcar las clases y métodos de prueba.
- Resultados de las pruebas: JUnit proporciona una serie de formatos para los resultados de las pruebas, incluyendo texto, HTML y XML.

Cómo utilizar JUnit

Para utilizar JUnit, primero debe instalarlo en su sistema. JUnit está disponible como un paquete JAR que se puede descargar desde el sitio web de JUnit.

Una vez instalado JUnit, puede comenzar a escribir pruebas unitarias. Para escribir una prueba unitaria, debe crear una clase que extienda la clase `TestCase` de JUnit. La clase debe tener métodos anotados con la anotación `@Test`.

Por ejemplo, el siguiente código muestra una prueba unitaria simple que verifica que el método `sum()` de una clase `NumberUtils` devuelva el valor correcto:

```
public class NumberUtilsTest extends TestCase {

    @Test
    public void testSum() {
        assertEquals(5, NumberUtils.sum(2, 3));
    }
}
```

Para ejecutar las pruebas unitarias, puede utilizar la herramienta `junit` proporcionada por JUnit. Por ejemplo, para ejecutar las pruebas unitarias de la clase `NumberUtilsTest`, puede ejecutar el siguiente comando:

```
java -cp junit-4.13.2.jar:. NumberUtilsTest
```

JUnit ofrece una serie de beneficios a los desarrolladores, incluyendo:

- Ayuda a mejorar la calidad del código: Las pruebas unitarias ayudan a detectar errores en el código temprano en el proceso de desarrollo.
- Reduce el tiempo de desarrollo: Las pruebas unitarias pueden ayudar a reducir el tiempo de desarrollo al detectar errores temprano.
- Facilita la refactorización: Las pruebas unitarias pueden ayudar a facilitar la refactorización del código al garantizar que el código modificado siga funcionando correctamente.

Bibliografia

Espresso. (s/f). Android Developers. Recuperado el 24 de noviembre de 2023, de https://developer.android.com/training/testing/espresso?hl=es-419

Wikipedia contributors. (s/f). Espresso (framework). Wikipedia, The Free Encyclopedia. https://es.wikipedia.org/w/index.php?title=Espresso_(framework)&oldid=146292908

Wikipedia contributors. (s/f). Espresso (framework). Wikipedia, The Free Encyclopedia. https://es.wikipedia.org/w/index.php?title=Espresso_(framework)&oldid=146292908