{desafío}
latam_

CURSO

TEST AUTOMATION ENGINEER
FORMACIÓN INTEGRAL

Objetivo General del Curso

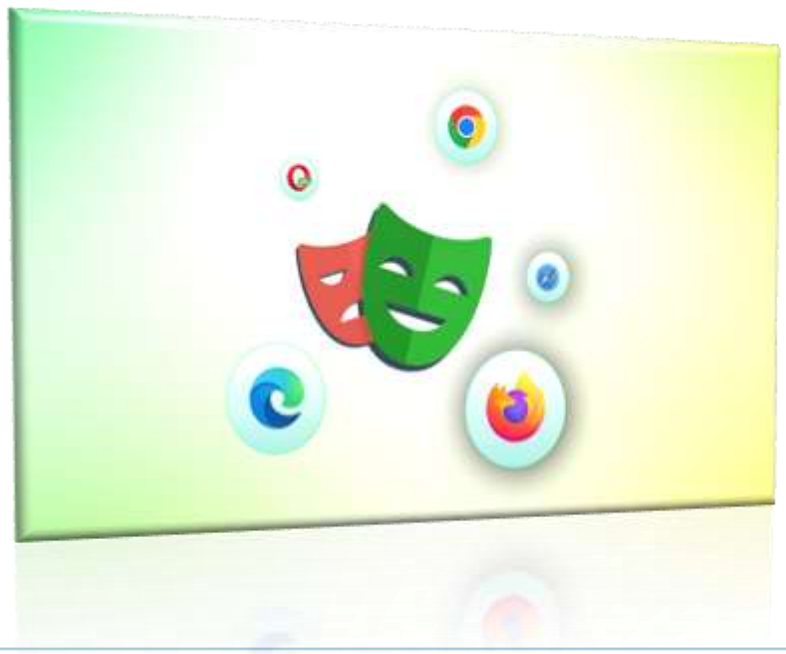
ANALIZAR LAS HERRAMIENTAS DE DISEÑO DE TEST AUTOMATION ENGINEER, DE ACUERDO A LAS APLICACIONES WEB, MÓVILES Y APIS.

Objetivo específico del Módulo

DETERMINAR HERRAMIENTAS DE PLAYWRIGHT Y PRUEBAS CON MÚLTIPLES NAVEGADORES, DE ACUERDO A LAS APLICACIONES WEB, MÓVILES Y APIS.

Contenidos

Módulo 7, HERRAMIENTAS DE PLAYWRIGHT Y PRUEBAS CON MÚLTIPLES NAVEGADORES



En el contexto del desarrollo web moderno, garantizar la calidad del software es una prioridad clave, y las herramientas de automatización de pruebas han cobrado una relevancia creciente. Una de las soluciones más poderosas y versátiles en esta área es Playwright, un framework de automatización de pruebas de código abierto desarrollado por Microsoft, diseñado específicamente para aplicaciones web. Playwright se ha consolidado como una herramienta robusta para la creación de pruebas de extremo a extremo (end-to-end), permitiendo a los desarrolladores simular interacciones reales de los usuarios en diferentes navegadores con gran precisión.

Su principal ventaja radica en la capacidad de automatizar tareas sobre múltiples motores de navegador —Chromium, Firefox y WebKit— de forma simultánea, lo que garantiza una cobertura completa del comportamiento de las aplicaciones en los principales entornos de usuario. Además, Playwright soporta múltiples lenguajes de programación, incluyendo JavaScript, TypeScript, Python, Java y .NET, lo cual lo convierte en una herramienta flexible, adaptable y fácilmente integrable en diversos flujos de trabajo de desarrollo y pruebas.

En este capítulo se abordarán los fundamentos de Playwright, sus características distintivas, la estructura básica de una prueba automatizada y las prácticas recomendadas para comenzar a trabajar con esta herramienta de forma efectiva.

Capítulo 1: FUNDAMENTOS DE PLAYWRIGHT

Playwright es una herramienta de automatización de pruebas web que permite interactuar con navegadores de forma programática para simular la experiencia de usuario. A diferencia de otras herramientas tradicionales como Selenium, Playwright fue diseñado desde su origen para trabajar con aplicaciones web modernas que utilizan tecnologías como JavaScript dinámico, Single Page Applications (SPA), y manejo de recursos asincrónicos.

Los fundamentos de Playwright se basan en tres pilares:

- **Automatización de navegadores modernos:** Playwright permite el control de navegadores como Chromium (Google Chrome), Firefox y WebKit (Safari) mediante una misma interfaz, sin necesidad de cambiar el código base. Esta compatibilidad se logra mediante un motor de ejecución que abstrae las diferencias entre navegadores y permite escribir pruebas una sola vez, ejecutándolas en múltiples entornos.

- **Soporte multiplataforma y multilenguaje**: Aunque Playwright se desarrolló principalmente en Node.js, también ofrece bindings para Python, Java y C#. Esto facilita su uso por parte de equipos con distintas competencias técnicas, integrándose fácilmente en diferentes ecosistemas tecnológicos.

- **Pruebas robustas y consistentes:** Una de las mayores fortalezas de Playwright es su capacidad para esperar automáticamente a que los elementos estén disponibles, evitando los típicos errores de sincronización. También permite interceptar peticiones de red, simular estados del navegador (como geolocalización o permisos), y trabajar con múltiples pestañas o contextos de usuario en paralelo, lo que lo hace ideal para pruebas complejas y de alto nivel.

Capítulo 2: INSTALACIÓN, SINTAXIS BÁSICA Y ARQUITECTURA

Instalación de Playwright

La instalación de Playwright es un proceso relativamente sencillo que se adapta a diferentes lenguajes y entornos de trabajo. En este apartado, nos centraremos en la instalación utilizando Node.js, que es la forma más común y extendida de trabajar con Playwright.

Requisitos previos

Antes de instalar Playwright, es necesario tener instalado:

- Node.js (versión 14 o superior): Se recomienda utilizar versiones LTS (Long Term Support).
- npm (Node Package Manager), que viene incluido con Node.js.
- Un editor de texto o IDE, como Visual Studio Code.

Instalación paso a paso

1.- Inicializar un nuevo proyecto Node.js

Abra una terminal y ejecute el siguiente comando para crear un archivo package.json:

```bash
npm init -y
```

2.- Instalar Playwright

Puede instalar Playwright junto con los navegadores necesarios usando:

```bash
npm install -D @playwright/test
npx playwright install
```

Este comando descarga los navegadores compatibles (Chromium, Firefox, WebKit) que se ejecutarán de forma aislada, sin depender del navegador instalado en el sistema.

3.- Verificación de instalación

Para comprobar que Playwright está correctamente instalado, puede ejecutar una prueba automática:

```bash
npx playwright test
```

Esto generará una estructura de carpetas predeterminada (tests/, playwright.config.ts, etc.) y ejecutará cualquier prueba de ejemplo que se haya creado.

Alternativas por lenguaje

- Python:

```bash
pip install playwright
playwright install
```

Java (a través de Maven o Gradle) y .NET también tienen sus propios procedimientos, aunque el principio es el mismo: instalar la librería y luego ejecutar la descarga de navegadores.

Sintaxis básica de Playwright

Playwright ofrece una API intuitiva y expresiva que permite crear pruebas legibles y robustas.

A continuación, se explica la estructura básica de un script de prueba en JavaScript/TypeScript usando el framework @playwright/test.

Estructura de una prueba

```javascript
import { test, expect } from '@playwright/test';

test('Mi primera prueba con Playwright', async ({ page }) => {
  // Ir a una página web
  await page.goto('https://example.com');

  // Verificar el título de la página
  await expect(page).toHaveTitle('Example Domain');

  // Hacer clic en un enlace
  await page.click('text=More information');

  // Verificar la URL
  await expect(page).toHaveURL(/iana.org/);
});
```

Componentes clave:

- test(): Define una prueba individual con un nombre descriptivo.
- page: Representa una pestaña del navegador y permite simular acciones del usuario.
- await: La mayoría de las acciones son asincrónicas, por lo que se requiere await para esperar su finalización.
- expect(): Se utiliza para hacer afirmaciones sobre el estado de la página, como su título, contenido o URL.

Selectores

Playwright admite múltiples tipos de selectores:

- Texto visible: 'text=Iniciar sesión'
- CSS: 'button.submit'
- XPath: '//button[text()="Enviar"]'
- Selectores compuestos: 'form >> text=Registrar'
- Selectores de rol (accesibilidad): 'role=button[name="Aceptar"]'

Arquitectura de Playwright

Playwright se basa en una arquitectura moderna diseñada para maximizar la confiabilidad, velocidad y paralelismo en las pruebas automatizadas. Su diseño interno permite controlar múltiples navegadores, contextos y páginas simultáneamente, facilitando pruebas avanzadas como simulaciones de múltiples usuarios o navegación con datos compartidos.

Componentes principales de la arquitectura

- **Cliente de pruebas**: Es el script de pruebas escrito por el desarrollador, que ejecuta comandos y define flujos de interacción con la aplicación. Se ejecuta en Node.js y utiliza la API de Playwright.

- **Servidor del navegador**: Playwright lanza versiones empaquetadas de Chromium, Firefox y WebKit, ejecutadas en segundo plano. Estos navegadores están controlados mediante protocolos de comunicación como CDP (Chrome DevTools Protocol) o APIs propias, dependiendo del motor.

- **Contextos de navegador (browser contexts):** Un contexto es una sesión aislada dentro de un navegador. Se comporta como una ventana separada con su propio almacenamiento local, cookies y caché. Esto permite simular múltiples usuarios o entornos sin tener que lanzar un nuevo navegador.

- **Páginas (pages):** Cada contexto puede tener múltiples páginas o pestañas, que representan las interfaces visuales sobre las cuales se interactúa. Las acciones como page.goto(), page.click(), page.fill() o page.screenshot() se ejecutan sobre estas páginas.

- **Test Runner (Playwright Test):** Es el entorno que organiza y ejecuta las pruebas. Soporta ejecución paralela, fixtures (datos y contextos preconfigurados), hooks (beforeEach, afterEach), informes en HTML, y más.

- **Grabación de acciones e inspección**: Playwright incluye herramientas visuales como npx playwright codegen para grabar interacciones de usuario y generar automáticamente código de prueba. También cuenta con playwright inspector, una interfaz visual que permite pausar y depurar pruebas paso a paso.

Resumen de ventajas arquitectónicas

- Ejecución paralela de pruebas.
- Aislamiento total entre contextos.
- Simulación precisa del entorno del usuario (ubicación, dispositivos, permisos).
- Captura automática de trazas, capturas de pantalla y videos.
- Bajo mantenimiento gracias al manejo automático de esperas.

Capítulo 3: COMPARACIÓN CON CYPRESS

Comparación entre Playwright y Cypress

Playwright y Cypress son dos de las herramientas de automatización de pruebas end-to-end más populares y modernas en el ecosistema web. Ambas han sido desarrolladas con el propósito de ofrecer una experiencia de testing más robusta, moderna y confiable que las herramientas tradicionales como Selenium. Aunque tienen objetivos similares, difieren significativamente en su enfoque, arquitectura, funcionalidades y limitaciones.

A continuación, se presenta una comparación profunda entre ambos frameworks.

1.- Compatibilidad con Navegadores

Característica	Playwright	Cypress
Navegadores compatibles	Chromium, Firefox, WebKit (Safari)	Chromium y derivados (Chrome, Edge), Firefox (experimental)
Soporte para Safari	Sí (WebKit)	No (ni siquiera experimental)
Control de múltiples navegadores	Sí (simultáneo y por código)	Limitado, no simultáneo

Ventaja: Playwright, por su cobertura multiplataforma que incluye Safari, lo que es ideal para pruebas cross-browser completas.

2. Lenguajes de programación compatibles

Característica	Playwright	Cypress
Lenguaje principal	JavaScript / TypeScript	JavaScript
Otros lenguajes soportados	Python, Java, .NET	No oficialmente

Ventaja: Playwright, por permitir su uso en distintos lenguajes, facilitando su adopción en equipos con variadas competencias técnicas.

3. Arquitectura de ejecución

Característica	Playwright	Cypress
Ejecución dentro del navegador	No (control remoto del navegador)	Sí (se ejecuta en el mismo ciclo de eventos que la app)
Aislamiento de contexto	Sí, múltiples contextos aislados	No, se limita a una sola ventana/pestaña por prueba
Soporte para múltiples pestañas	Sí	No
Automatización de descargas, carga de archivos, etc.	Completa	Parcial, con trabajo adicional

Ventaja: Playwright, por su capacidad de manejar múltiples contextos, pestañas, descargas y más, lo que permite pruebas más avanzadas y realistas.

4. Sincronización y esperas

Característica	Playwright	Cypress
Esperas automáticas	Sí, integradas por defecto	Sí, pero a veces requiere configuración
Manejo de asincronía	Robusto y explícito (uso de await)	Implicitamente manejado por el motor interno
Retries automáticos en comandos	Sí	Sí

Ventaja: Empate. Ambos frameworks manejan la sincronización de forma eficaz, aunque con estilos diferentes.

5. Depuración y herramientas visuales

Característica	Playwright	Cypress
Herramienta de inspección visual	playwright inspector	Cypress Test Runner (interactivo)
Grabador de código automático	Sí (codegen)	Parcial, a través de plugins
Informes visuales	Sí (HTML, videos, trazas)	Sí (nativo y con plugins)

Ventaja: Depende del caso. Cypress tiene un entorno visual más amigable para principiantes; Playwright ofrece más profundidad para debugging avanzado.

6. Integración con CI/CD y escalabilidad

Característica	Playwright	Cypress
Soporte en CI/CD	Sí (Jenkins, GitHub Actions, GitLab, etc.)	Sí
Ejecución paralela	Sí (local y distribuida)	Sí (aunque algunas funciones son exclusivas de Cypress Cloud)
Headless mode	Sí	Sí

Ventaja: Playwright, especialmente por su ejecución paralela local sin necesidad de servicios pagos.

7. Licencia y comunidad

Característica	Playwright	Cypress
Licencia	Apache 2.0 (gratis, de código abierto)	MIT (gratuita) + funcionalidades avanzadas bajo pago
Comunidad	En crecimiento, fuerte apoyo de Microsoft	Muy consolidada, gran comunidad de frontend
Integración con herramientas de terceros	Buena (allure, Jest, Docker, etc.)	Excelente, con plugins específicos para React, Vue, etc.

Ventaja: Cypress, en términos de comunidad y plugins; Playwright está creciendo rápidamente.

Ambas herramientas ofrecen un entorno moderno, potente y amigable para el testing de aplicaciones web. Sin embargo, Playwright se destaca por su capacidad multiplataforma, soporte para múltiples lenguajes, manejo de contextos aislados y funciones avanzadas de automatización, lo que lo convierte en una solución ideal para pruebas complejas, aplicaciones con múltiples sesiones, o donde se requiere compatibilidad con Safari.

Por otro lado, Cypress destaca por su facilidad de uso, experiencia visual interactiva y comunidad activa, lo que lo hace muy atractivo para equipos de frontend y proyectos que no requieren pruebas tan avanzadas o distribuidas.

Capítulo 4: EJECUCIÓN EN DISTINTOS NAVEGADORES (CHROMIUM, FIREFOX, WEBKIT).

Una de las principales fortalezas de Playwright frente a otras herramientas de automatización es su capacidad nativa para ejecutar pruebas en los tres principales motores de navegador: Chromium, Firefox y WebKit. Esto permite realizar pruebas verdaderamente cross-browser, cubriendo una amplia gama de entornos reales en los que los usuarios pueden interactuar con una aplicación web.

¿Qué significa esto en la práctica?

Cada motor de navegador representa una familia de navegadores distintos:

- **Chromium:** utilizado por Google Chrome, Microsoft Edge, Brave, Opera, entre otros.
- **Firefox:** el motor Gecko, específico del navegador Mozilla Firefox.
- **WebKit:** el motor usado por Safari (en macOS y dispositivos iOS), y también por algunos navegadores embebidos.

Playwright automatiza estos motores directamente, sin depender de WebDriver, lo que le otorga mayor precisión, velocidad y control sobre el entorno de ejecución.

Ventajas de la ejecución cross-browser en Playwright

- Cobertura realista del entorno del usuario: permite verificar que una aplicación funcione correctamente no solo en Chrome, sino también en Safari (a través de WebKit) y Firefox.
- Detección temprana de errores específicos por navegador: muchos bugs de compatibilidad aparecen solo en ciertos navegadores. Probar en los tres evita sorpresas en producción.
- Control total del navegador: al manejar directamente los motores, Playwright permite acciones avanzadas como modificar la geolocalización, simular condiciones de red o interceptar peticiones HTTP.
- Pruebas en paralelo: se pueden ejecutar pruebas en múltiples navegadores simultáneamente, lo cual ahorra tiempo y mejora la eficiencia del proceso de QA.

Ejemplo de ejecución en distintos navegadores

Playwright permite especificar fácilmente en qué navegador correr una prueba. Aquí un ejemplo básico en JavaScript/TypeScript:

```javascript
const { chromium, firefox, webkit } = require('playwright');

(async () => {
  for (const browserType of [chromium, firefox, webkit]) {
    const browser = await browserType.launch();
    const context = await browser.newContext();
    const page = await context.newPage();
    await page.goto('https://example.com');
    await page.screenshot({ path: `screenshot-${browserType.name()}.png` });
    await browser.close();
  }
})();
```

Este script ejecuta la misma prueba en los tres navegadores, toma una captura de pantalla en cada uno y la guarda con un nombre identificador. La misma lógica puede aplicarse para pruebas completas usando frameworks de testing como Jest o Test Runner propio de Playwright.

Automatización simultánea

Playwright también permite definir proyectos dentro de su configuración (playwright.config.ts o .js) para ejecutar las pruebas en varios navegadores en paralelo de forma automática:

```ts
import { defineConfig, devices } from '@playwright/test';

export default defineConfig({
  projects: [
    {
      name: 'Chromium',
      use: { browserName: 'chromium' },
    },
    {
      name: 'Firefox',
      use: { browserName: 'firefox' },
    },
    {
       name: 'WebKit',
       use: { browserName: 'webkit' },
     },
  ],
});
```

Al correr npx playwright test, las pruebas se ejecutarán en los tres navegadores automáticamente.

Consideraciones adicionales

- **Compatibilidad móvil:** Playwright también permite emular dispositivos móviles en cada motor, lo cual es esencial para pruebas responsive.

- **WebKit y Safari**: aunque Playwright usa WebKit y no Safari directamente, el resultado de las pruebas es altamente representativo del comportamiento real en Safari, lo que cubre una necesidad crítica para quienes desarrollan en Apple.

- **Instalación ligera:** al instalar Playwright, los binarios de los navegadores se descargan automáticamente, por lo que no es necesario tenerlos instalados previamente en el sistema.

La capacidad de ejecutar pruebas en Chromium, Firefox y WebKit de forma sencilla, controlada y eficiente, posiciona a Playwright como una herramienta altamente robusta para pruebas de compatibilidad. Esta funcionalidad permite a los equipos garantizar que su aplicación ofrezca una experiencia consistente y libre de errores en todos los navegadores principales, mejorando la calidad general del software y la confianza del usuario final.

Capítulo 5: PRUEBAS PARALELAS Y ASINCRONÍA

Playwright está diseñado con un enfoque moderno y altamente eficiente para ejecutar pruebas automatizadas, lo que incluye dos pilares fundamentales: la ejecución en paralelo y el manejo de asincronía. Ambos aspectos están profundamente integrados en su arquitectura y permiten escalar la automatización sin sacrificar precisión ni estabilidad.

Ejecución de Pruebas en Paralelo

Una de las características más potentes de Playwright es su capacidad de ejecutar pruebas en paralelo, lo que reduce drásticamente los tiempos de ejecución en suites grandes y mejora la eficiencia en entornos de integración continua (CI/CD).

¿Cómo funciona?

Playwright ejecuta pruebas en workers independientes, cada uno con su propio navegador, contexto y página, lo que asegura el aislamiento completo entre pruebas. Esto previene efectos colaterales o interferencias entre los casos de prueba.

Configuración de pruebas paralelas

En playwright.config.ts, puedes definir cuántos workers deseas usar:

```ts
import { defineConfig } from '@playwright/test';

export default defineConfig({
  workers: 4, // Ejecutar 4 pruebas en paralelo
});
```

También puedes permitir que Playwright maneje automáticamente el número de workers según los núcleos del sistema (workers: undefined).

Ejemplo: ejecución paralela en múltiples navegadores

Playwright puede ejecutar las mismas pruebas en distintos navegadores simultáneamente mediante la funcionalidad de proyectos:

```ts
projects: [
  { name: 'Chromium', use: { browserName: 'chromium' } },
  { name: 'Firefox', use: { browserName: 'firefox' } },
  { name: 'WebKit', use: { browserName: 'webkit' } },
]
```

Cuando ejecutas npx playwright test, Playwright corre las pruebas en los tres navegadores en paralelo.

Beneficios de la ejecución paralela

- Reducción de tiempos de ejecución en suites extensas.
- Escalabilidad horizontal en CI/CD.
- Aislamiento robusto, al no compartir estado entre pruebas.
- Compatible con sharding (división de pruebas entre múltiples máquinas o contenedores).

Asincronía en Playwright

Todas las interacciones con el navegador en Playwright son asincrónicas, lo que significa que cada acción devuelve una promesa que debe resolverse antes de continuar. Esto permite una ejecución más controlada, precisa y libre de errores.

¿Por qué es importante?

En entornos web modernos, muchas acciones como cargar una página, esperar un selector o recibir una respuesta de red no ocurren instantáneamente. Playwright maneja estos eventos de forma asincrónica y automática, previniendo condiciones de carrera o tiempos muertos innecesarios.

Uso de async/await

Todas las funciones principales de Playwright requieren el uso de async/await:

```ts
const { chromium } = require('playwright');

(async () => {
  const browser = await chromium.launch();
  const page = await browser.newPage();
  await page.goto('https://example.com');
  await page.click('text=Iniciar sesión');
  await browser.close();
})();
```

Cada acción espera que la anterior se complete exitosamente antes de continuar, garantizando sincronización precisa con el navegador.

Esperas automáticas (auto-waiting)

Playwright incorpora esperas inteligentes automáticas. Por ejemplo:

await page.click('selector') esperará automáticamente a que el selector esté:

- presente en el DOM,
- visible,
- habilitado,
- y listo para recibir un clic.

Esto elimina la necesidad de waits manuales como setTimeout, y evita errores intermitentes comunes en otras herramientas de automatización.

Ejecución asincrónica con múltiples acciones

Es posible lanzar múltiples promesas en paralelo (por ejemplo, cuando se espera una navegación tras una acción):

```ts
await Promise.all([
  page.waitForNavigation(),
  page.click('text=Continuar')
]);
```

Este patrón garantiza que la prueba no continúe hasta que ambas acciones se completen, lo que mejora la robustez de los tests.

Playwright combina un modelo asincrónico moderno con un sistema altamente eficiente de ejecución paralela, ofreciendo una solución escalable, rápida y precisa para la automatización de pruebas web.

- La ejecución paralela permite distribuir cargas, acelerar los tiempos y escalar horizontalmente.

- La asincronía integrada, basada en promesas y await, garantiza sincronización natural con la dinámica de las aplicaciones web actuales.

- En conjunto, estas capacidades hacen de Playwright una herramienta ideal para equipos que buscan eficiencia, estabilidad y facilidad de mantenimiento en pruebas automatizadas.