### BUENAS PRACTICAS CON PLAYWRIGHT

### ****ESTRUCTURA Y ORGANIZACIÓN DEL CÓDIGO****

**Divide tus pruebas en archivos y carpetas lógicas**: Organiza tus tests en carpetas y archivos basados en funcionalidades, módulos o características de tu aplicación. Esto facilita la navegación y el mantenimiento del código.

**Usa *test.describe* para agrupar pruebas relacionadas**: test.describe te permite agrupar tests relacionados, lo que mejora la organización y te permite aplicar configuraciones específicas a un grupo de pruebas.

### 2. ****USO DE FIXTURES Y CONFIGURACIÓN GLOBAL****

**Configura Fixtures reutilizables**: Utiliza fixtures para establecer configuraciones comunes, como la inicialización de datos o la configuración del entorno antes de cada prueba.

Puedes usar **test.beforeEach** y **test.afterEach** para configurar y limpiar datos antes y después de cada prueba.

**Configura un archivo playwright.config.ts centralizado**: Define configuraciones globales como timeouts, reintentos, navegadores, y más en este archivo. Esto asegura consistencia en todas las pruebas.

### 3. ****ESPERA EXPLICITA (****AWAIT****)****

**Evita esperas fijas (**waitForTimeout**)**: Es mejor utilizar esperas explícitas **(expect con toBeVisible, toHaveText, etc.)** en lugar de esperas fijas, ya que las esperas fijas pueden hacer que las pruebas sean más lentas e inestables.

**Utiliza las esperas automáticas de Playwright**: Playwright automáticamente espera hasta que el elemento esté listo antes de interactuar con él, pero también puedes utilizar **page.waitForSelector** o **page.waitForLoadState** para manejar situaciones más complejas.

### 4. ****SELECTORES****

**PREFIERE SELECTORES ROBUSTOS Y SEMÁNTICOS**: Usa selectores que sean estables y no dependan de cambios frecuentes en la estructura HTML. Selectores como [data-test-id="login-button"] o texto visible (text=Log In) son más mantenibles que selectores basados en la estructura del DOM (div > span > button).

**EVITA SELECTORES EXCESIVAMENTE COMPLEJOS**: Mantenlos simples y legibles, lo que facilita su mantenimiento.

### 5. ****MANEJO DE DATOS Y ESTADO****

****AISLA PRUEBAS**:** Asegúrate de que cada prueba pueda ejecutarse de manera independiente, sin depender del estado dejado por otra prueba.

Esto incluye no compartir datos de prueba entre tests a menos que sea absolutamente necesario.

**USA DATOS CONSISTENTES Y LIMPIABLES**: Si tus pruebas necesitan crear datos (como usuarios o registros en una base de datos), asegúrate de que esos datos sean consistentes y puedan ser limpiados o reseteados después de cada prueba.

### 6. ****OPTIMIZACIÓN Y RENDIMIENTO****

****EJECUTA PRUEBAS EN PARALELO**:** Configura tus pruebas para que se ejecuten en paralelo en diferentes procesos o navegadores cuando sea posible.

Playwright soporta la ejecución paralela de pruebas por defecto.

****REUTILIZA CONTEXTOS Y NAVEGADORES**:** Reutiliza instancias de navegadores y contextos para pruebas que no necesitan aislamiento total, lo que puede mejorar el rendimiento.

### 7. ****BUENA GESTIÓN DE ERRORES****

**USA TRY-CATCH CUANDO SEA NECESARIO**: Aunque Playwright maneja automáticamente muchos errores, en casos especiales puedes querer capturar y manejar errores específicos para proporcionar mensajes más claros o realizar limpiezas adicionales.

**HAZ CAPTURAS DE PANTALLA O LOGS EN FALLOS**: Configura Playwright para tomar capturas de pantalla o guardar logs en caso de fallo de las pruebas.

Esto es útil para depurar fallos en entornos CI.

### 8. ****CONFIGURACIÓN DE RETRIES Y TIMEOUTS****

****CONFIGURA** RETRIES **Y** TIMEOUT **DE MANERA ADECUADA**:** Ajusta los reintentos (retries) y tiempos de espera (timeout) en función de tu entorno. En CI, puedes configurar más reintentos debido a la inestabilidad inherente de estos entornos.

### 9. ****CUIDADO CON LAS DEPENDENCIAS EXTERNAS****

****SIMULA O INTERCEPTA SOLICITUDES DE RED**:** Evita depender de servicios externos en tus tests. Utiliza Playwright para interceptar y simular respuestas de red (page.route), lo que hace que las pruebas sean más rápidas y fiables.

****DESHABILITA ANIMACIONES Y TIEMPOS DE ESPERA**:** Desactiva animaciones y otros elementos visuales que puedan afectar la estabilidad de los tests.

### 10. ****DOCUMENTACIÓN Y MANTENIMIENTO****

**DOCUMENTA PRUEBAS COMPLEJAS**: Si una prueba es particularmente compleja o tiene dependencias especiales, asegúrate de documentarla adecuadamente para que otros desarrolladores puedan entenderla y mantenerla.

**REVÍSALAS REGULARMENTE**: Revisa y actualiza tus pruebas regularmente para asegurarte de que siguen siendo relevantes y que no están probando funcionalidades obsoletas.