# Sesiones multiples

Esta es una excelente evolución para tu herramienta. Convertirla en una plataforma de *testing* "End-to-End" (E2E) para dos equipos es un salto de funcionalidad muy potente.

Para lograr esto, necesitarás una refactorización (un rediseño) de la arquitectura actual, ya que la aplicación está construida alrededor de una sola sesión de navegador.

Aquí tienes un plan paso a paso sobre cómo abordarlo, junto con las ideas de pruebas que solicitas.

1. **Plan de Implementación: Sesiones Múltiples**

El objetivo es modificar la aplicación para que, en lugar de gestionar un solo browser\_process, gestione un diccionario de sesiones (por ejemplo, "Equipo A" y "Equipo B").

1. **Paso 1: Modificar la GUI (El "Sidebar")**

Primero, el usuario necesita poder conectarse a dos equipos. El gui/ui\_sidebar.py es el lugar para esto.

1. **Doble Conexión:** Duplica los campos de IP/Usuario/Contraseña para tener "Equipo A" y "Equipo B".
2. **Doble Estado:** Duplica las etiquetas de estado (session\_status\_label) para mostrar el estado de A y B.
3. **Selector de Contexto:** Añade un CTkSegmentedButton en la parte superior del sidebar con las opciones "Equipo A" | "Equipo B". Esto le dirá al resto de la aplicación cuál es el equipo "activo" sobre el que se ejecutarán los tests individuales (como consultar módulos, alarmas, etc.).
4. **Paso 2: Refactorizar el Núcleo (app.py y session\_controller.py)**

Aquí es donde se maneja el estado de las sesiones.

1. **En gui/app.py:**
   * Reemplaza self.browser\_process = None por un diccionario:

Python

self.sessions = {

'A': {'ip': None, 'process': None, 'status': 'Desconectado'},

'B': {'ip': None, 'process': None, 'status': 'Desconectado'}

}

self.active\_session\_id = 'A' # Controlado por el SegmentedButton

1. **En logic/session\_controller.py:**
   * Modifica \_execute\_start\_browser y \_execute\_stop\_browser para que acepten un session\_id ('A' o 'B').
   * Necesitarás que tu BrowserManager.py registre los navegadores con un *alias* (alias=A, alias=B).
   * \_execute\_start\_browser ya no leerá la IP de la GUI, sino que la recibirá como parámetro (ip\_equipo\_A, user\_A, etc.).
2. **Paso 3: Refactorizar los Controladores y robot\_executor.py**

Todos los tests actuales (Monitorización, Alineación, etc.) deben saber a qué sesión (A o B) deben atacar.

1. **En logic/robot\_executor.py:**
   * La función principal \_run\_robot\_test debe aceptar un nuevo parámetro: session\_id.
   * Este session\_id se usará para construir el comando de Robot Framework y decirle a qué navegador conectarse (usando el alias del BrowserManager).
2. **En todos los demás controladores (monitoring\_controller.py, equipment\_controller.py, etc.):**
   * Cuando llamas a \_run\_robot\_test, ahora debes pasar el session\_id activo, que obtienes del app\_ref: session\_id = self.app\_ref.active\_session\_id.
3. **Nuevas Ideas de Pruebas (Tu Objetivo)**

Una vez completada la refactorización, puedes crear una nueva pestaña en la GUI (ej. "Pruebas E2E") que ejecute tests que controlen **ambas sesiones a la vez**.

Aquí tienes las ideas que mencionaste:

1. **Test de Latencia de Transmisión (Cálculo de Transmisión):**
   * **GUI:** Un botón "Iniciar Test de Latencia E2E".
   * **Test:** Un nuevo script de Robot que:
     1. Se conecta a la sesión A y la sesión B.
     2. (En A): Activa el Input 1 (usando Input Activation).
     3. (En A): Consulta el Registro Cronológico y extrae el *timestamp* exacto de la activación (ej. T1 = 08:00:17.380).
     4. (En B): Consulta el Registro Cronológico y busca el evento de "Recepción de Comando 1". Extrae su *timestamp* (ej. T2 = 08:00:17.450).
     5. (En B): Opcionalmente, verifica el trap SNMP de recepción (tpu1cNotifyCommandRx).
   * **Resultado:** El test calcula Latencia = T2 - T1 (ej. 70ms) y lo reporta a la GUI.
2. **Test de Verificación de Comandos E2E:**
   * **GUI:** Similar al test de correlación, pero E2E.
   * **Test:**
     1. (En A): Activa INPUT (1) CIRCUIT.
     2. (En B): Verifica que OUTPUT (1) CIRCUIT se activa.
     3. (En B): Verifica que el Registro Cronológico de B registra la recepción del comando.
     4. (En B): Verifica que se recibe el trap tpu1cNotifyCommandRx (si la MIB lo soporta).
3. **Test de Estrés y "Flapping" (Intermitencia):**
   * **GUI:** Un botón para iniciar un test de estrés de 5 minutos.
   * **Test:**
     1. (En B): Inicia un monitoreo de la lista de Alarmas.
     2. (En A): Usando el Planificador (Scheduler), ejecuta la activación y desactivación del Input 1 en un bucle rápido (ej. cada 2 segundos) durante 5 minutos.
   * **Resultado:** El test verifica que el Equipo B reporta las alarmas correctamente sin "flapping" (alarmas que aparecen y desaparecen erráticamente) o que el enlace se mantiene estable.

Este es un proyecto de desarrollo considerable, pero muy valioso. Te recomiendo empezar por el **Paso 1 (Sidebar)** y el **Paso 2 (Núcleo de Sesiones)**. Una vez que puedas conectarte y desconectarte de A y B independientemente, el resto será mucho más fácil.