**~~Parte 1: Independencia de Datos de Sesión~~**

1. **~~Crear un Modelo de Datos por Sesión:~~** ~~Modificar app.py para que ModernTestRunnerApp contenga un diccionario que almacene los~~ *~~datos~~* ~~de la GUI (listas de módulos, datos de configuración, etc.) de forma separada para 'A' y 'B'.~~
2. **~~Poblar el Modelo de Datos:~~** ~~Modificar process\_gui\_queue en app.py. Las tareas que devuelven datos (ej: modules\_display) ahora también deben incluir el session\_id que las originó. La cola guardará los datos en el diccionario de la sesión correspondiente (ej: self.session\_gui\_data['A']['modules\_list'] = ...).~~
3. **~~Crear Función de Refresco:~~** ~~Crear una nueva función en app.py, (ej: \_update\_gui\_from\_active\_session()). Esta función leerá el self.active\_session\_id y usará los datos del diccionario de esa sesión (self.session\_gui\_data['A'] o self.session\_gui\_data['B']) para actualizar todos los widgets de la GUI.~~
4. **~~Conectar Cambio de Contexto:~~** ~~Modificar la función set\_active\_context en app.py para que, además de cambiar el active\_session\_id, llame a la nueva función de refresco y así "redibuje" la GUI con los datos de la sesión seleccionada.~~

**~~Parte 2: Independencia del Scheduler~~**

1. **~~Modificar GUI del Scheduler:~~** ~~En gui/ui\_tab\_scheduler.py, añadir un nuevo CTkComboBox o CTkSegmentedButton en el frame de "Crear Tarea" para seleccionar "Sesión A" o "Sesión B".~~
2. **~~Guardar Sesión en la Tarea:~~** ~~En logic/scheduler\_controller.py, modificar \_add\_task\_to\_sequence para que lea este nuevo selector y guarde el session\_id elegido dentro del diccionario de la tarea.~~
3. **~~Actualizar Lógica de Ejecución:~~** ~~En logic/scheduler\_controller.py, modificar el bucle de ejecución de tareas (\_execute\_next\_task o similar) para que, en lugar de usar self.app\_ref.active\_session\_id, use el session\_id específico guardado en~~ *~~cada tarea~~* ~~al ejecutarla.~~
4. **~~Desacoplar Botones del Scheduler:~~** ~~En app.py, modificar run\_button\_state para que la activación/desactivación de los botones "Ejecutar Secuencia" y "Detener Secuencia" dependa~~ *~~únicamente~~* ~~de si hay una tarea corriendo (is\_running), y~~ *~~no~~* ~~del session\_active (el contexto A/B seleccionado).~~
   * Necesito incorporar una nueva funcionalidad en la GUI que he desarrollado con la que se pueden realizar pruebas automatizadas mediante la web de configuración de un equipo TPU-1 de la empresa ZIV.   
     Necesito poder verificar que los tests que se ejecutan en el planificador de tareas sean correctos y que el equipo se comporta correctamente. Esto lo quiero hacer a través de la captura de traps que ya he conseguido llevar a cabo (la tarea de Verificar Traps SNMP). La idea sería por ejemplo (es el caso por el que quiero empezar y que a la empresa más le interesa): poder activar inputs mediante uno de los equipos (indicaríamos mediante el selector de contexto de qué sesión se trata), que el equipo envíe ordenes Y QUE PODAMOS VALIDAR QUE ESAS ORDENES HAYAN SALIDO MEDIANTE LOS TRAPS CAPTURADOS. La idea sería hacer algo (tipo un CSV u otra sección en la que podamos verificar mediante traps que estos inputs han sido realmente activados).

Me gustaría también poder llevar más allá estas validaciones para realizar pruebas más complejas en las que los dos equipos se vean involucrados.  
Cual sería tu plan de acción? ¿Cual sería la mejor forma de implementar algo así?  
Te adjunto el repositorio de Git donde podrás ver todos los archivos de mi proyecto.  
Gracias

## Plan de Acción

**~~Fase 1: Modificar la GUI (ui\_tab\_scheduler.py)~~**

1. **~~Crear el Campo de OID:~~** ~~Cuando el usuario añade la tarea "Verificar Traps SNMP" a la lista de la secuencia, necesitas que la GUI muestre un campo de entrada de texto (un QLineEdit o similar)~~ *~~específicamente para esa tarea~~*~~.~~
2. **~~Lógica de la GUI:~~**
   * ~~La lista de tareas seleccionadas (supongo que es un QListWidget o QTableWidget) tendrá que ser un poco más inteligente.~~
   * ~~Si el ítem añadido es "Verificar Traps SNMP", habilita/muestra un campo de texto al lado o en una sección de "Parámetros de Tarea" para que el usuario escriba el OID.~~
   * ~~Si es cualquier otra tarea, ese campo de OID no es necesario o está deshabilitado.~~

**~~Fase 2: Modificar el Guardado de Secuencia (scheduler\_controller.py)~~**

1. **~~Enriquecer el JSON:~~** ~~Cuando el usuario pulsa "Guardar secuencia de tareas", tu scheduler\_controller (en su función save\_schedule o similar) debe hacer lo siguiente:~~
   * ~~Recorre la lista de tareas de la GUI.~~
   * ~~Si la tarea es "Verificar Traps SNMP", debe leer el texto del QLineEdit asociado y guardarlo dentro de la estructura de esa tarea en el JSON.~~

~~Por ejemplo, el JSON guardado se vería así:~~

~~[~~

~~{~~

~~"task\_name": "SNMP\_ActivacionDesactivacionInputs.robot",~~

~~"variables": {~~

~~"INPUT\_OID": ".1.3.6.1.4.1.27419.1.2.3.1.5.1"~~

~~}~~

~~},~~

~~{~~

~~"task\_name": "Verificar Traps SNMP",~~

~~"variables": {~~

~~"EXPECTED\_OID": ".1.3.6.1.4.1.27419.1.2.8.1.1.0" // <-- ¡Aquí está el OID de la GUI!~~

~~}~~

~~},~~

~~{~~

~~"task\_name": "OtroTest.robot",~~

~~"variables": {}~~

~~}~~

~~]~~

**~~Fase 3: Modificar la Ejecución de Secuencia (scheduler\_controller.py)~~**

~~Aquí está el cambio principal en la lógica. El controlador ya no solo "ejecuta" tareas, sino que también "interpreta" algunas de ellas.~~

1. **~~Registro de Tiempo:~~** ~~El scheduler\_controller necesita saber~~ *~~cuándo~~* ~~se ejecutan las acciones.~~
   * ~~Al ejecutar una tarea de~~ *~~acción~~* ~~(como SNMP\_ActivacionDesactivacionInputs.robot), debe guardar un timestamp: T\_inicio\_accion = datetime.now().~~
2. **~~Detectar la Tarea de Verificación:~~** ~~Cuando el bucle de ejecución llega a la tarea "task\_name": "Verificar Traps SNMP":~~
   * **~~No la envía al robot\_executor~~**~~.~~
   * ~~En su lugar, activa la nueva lógica de validación.~~
3. **~~Lógica de Validación (dentro del scheduler\_controller o en un ValidationController):~~**
   * ~~Lee el OID a verificar desde task["variables"]["EXPECTED\_OID"].~~
   * ~~Define la ventana de tiempo. Lo más lógico es buscar desde el inicio de la última tarea de acción hasta el momento actual: (T\_inicio\_accion, datetime.now()).~~
   * ~~Llama al db\_handler.py con una nueva función, por ejemplo: found = db.check\_for\_trap(oid=EXPECTED\_OID, start\_time=T\_inicio\_accion, end\_time=datetime.now()).~~
4. **~~Reportar Resultado:~~**
   * ~~La función db.check\_for\_trap simplemente devuelve True o False.~~
   * ~~Si found == True, el scheduler\_controller marca la tarea "Verificar Traps SNMP" como~~ **~~PASS~~** ~~(✅) en la GUI.~~
   * ~~Si found == False, la marca como~~ **~~FAIL~~** ~~(❌) y, opcionalmente, podría detener la secuencia.~~

# Doble listener y asignación de nombre csv automática

## Gemini: “Mejoras GUI para Receptor SNMP”

Necesito incorporar una nueva funcionalidad en la GUI que he desarrollado con la que se pueden realizar pruebas automatizadas mediante la web de configuración de un equipo TPU-1 de la empresa ZIV. Esto lo hago manejando tests .robot que trabajan con las librerías Selenium y RobotFramework.

He conseguido implementar un planificador de tareas en el que puedo programar tanto tests de robot como un tipo de tarea ("Verificar Traps SNMP"), con el que puedo verificar aportando como entrada un OID concreto del trap que quiero verificar. Seguidamente se me crea un informe de la validación correspondiente y la evidencia de los traps encontrados con ese OID. A parte, TODOS LOS TRAPS (no solo los que coinciden con el OID) encontrados en la secuencia de tareas ejecutado se guardan en una base de datos de un archivo .db.

Me gustaría mejorar mi GUI implementando dos mejoras:

1. Poder Guardar más de un archivo .csv cada vez que se ejecuta la tarea "Verificar Traps SNMP Nuevos". Y que archivo csv que se cree se le asigne como nombre el OID que se espera encontrar. Si se repitiera la comprobación con el mismo OID, se tendría que añadir algun término que diferencie los archivos (un timestamp o una numeración).

2. Quiero realizar una mejora sustancial en la manera en la que recibo hasta ahora traps medante SNMP. Me gustaría poder escoger en que sesión ejecutar el listener Receptor de Traps. Hasta ahora solo es posible ejecutarlo EN UNA SOLA DIRECCIÓN IP .

Quiero modificar la estructura de la pestaña de Receptor de Traps. Quiero:

- Eliminar la entrada de la dirección IP del listener y que se asigne de forma automática

la propia del dispositivo (lo que haríamos dandole al boton de Usar IP Local). (esto dime si te parece una buena idea o si tiene algun inconveniente).

- Poder escoger el puerto asignado a cada listener de cada sesión. Tener un selector en la pestaña de "Receptor de Traps" que me permita escoger entre Receptor de Traps de la sesión "A" y de la sesión "B".

Si consideras que hay alguna forma más optima de implementar esto, hazmelo saber.

* **Eliminar la IP Manual:** ¡Sí, hazlo! Para un *listener* (receptor), lo ideal no es usar una IP local específica (como 192.168.1.10), sino enlazarlo a la dirección 0.0.0.0. Esto significa "escuchar en todas las interfaces de red" (Ethernet, Wi-Fi, etc.). Es la opción más simple y robusta. El botón "Usar IP Local" ya no es necesario.
* **Selector de Sesión (A/B):** Es la forma correcta de manejar dos listeners. Crearemos un botón segmentado para cambiar entre la configuración y el visor de "Listener A" y "Listener B".
* **Puerto por Sesión:** Absolutamente. Cada listener debe tener su propio puerto para evitar conflictos. Les daremos puertos por defecto distintos (ej. 171 y 172).

Implementaremos esto creando dos "juegos" de widgets (puerto, botones, estado, visor de traps) y mostraremos solo el juego correspondiente a la sesión seleccionada en el botón segmentado.

**2. csv.csvcsv Mejora de Informes CSV (Petición 1)**

También es una gran mejora. Actualmente, el archivo verification\_report.csv se sobrescribe. Lo cambiaremos para que, en lugar de un archivo global, la tarea "Verificar Traps SNMP Nuevos" genere un **archivo .csv único por cada vez que se ejecuta**.

El nombre del archivo se basará en el OID y la fecha/hora para garantizar que sea único, por ejemplo: 1\_3\_6\_1\_4\_1\_x\_x\_20251029\_143000.csv.

La pestaña "Informe Verificación" que ya tienes funcionará perfectamente para abrir y visualizar cualquiera de estos nuevos archivos generados.

# Tarea “Verificación de Traps SNMP Nuevos” -> Poder escoger sesión en la que se verifican traps.

## Gemini: “”

Ahora me gustaría quedo desde el Scheduler sea capaz también de poder escoger la sesión del listener en la que quiero que se verifiquen los traps SNMP. Me gustaría que en el nombre del archivo de los .csv creados apareciera la sesión en la que se verificaron esos traps.