necesito ganar dinero en mis tiempos libres haciendo dashboard en power bi, donde puedo realizar erso?





2 vídeos

¡Crear dashboards en Power BI es una excelente manera de ganar dinero en tus tiempos libres! Aquí tienes algunas ideas para empezar:

1. **Freelancing en plataformas en línea**: Sitios como Upwork y Fiverr tienen una gran demanda de expertos en Power BI. Puedes ofrecer tus servicios creando dashboards personalizados para empresas o individuos.
2. **Cursos y tutoriales**: Si tienes experiencia, podrías crear y vender cursos en plataformas como Udemy o Coursera. También puedes compartir tutoriales en YouTube para atraer clientes potenciales.
3. **Redes sociales y networking**: Usa LinkedIn para mostrar tus proyectos y conectar con empresas que necesiten servicios de visualización de datos. También puedes unirte a grupos de Power BI en redes sociales para encontrar oportunidades.
4. **Consultoría para pequeñas empresas**: Muchas pequeñas empresas necesitan ayuda para analizar sus datos. Ofrece tus servicios directamente a negocios locales.
5. **Colaboración con startups**: Las startups suelen buscar soluciones económicas para gestionar sus datos. Puedes ofrecerles dashboards que les ayuden a tomar decisiones estratégicas.

¿Te gustaría que te ayude a crear un perfil atractivo o a buscar más recursos para empezar? 😊

soy de chile, como puedo buscar trabajos por hora en mis tiempos libres, me gusta hacer dashboard en power bi y manejas los datos

¡Qué genial que te interese trabajar con Power BI y análisis de datos! Aquí tienes algunas opciones para buscar trabajos por hora en Chile:

1. **Plataformas de empleo**: Sitios como Computrabajo y Chiletrabajos tienen ofertas específicas para roles relacionados con Power BI. Puedes buscar trabajos por hora o proyectos freelance.
2. **Networking profesional**: Usa LinkedIn para conectar con empresas locales y mostrar tus habilidades. Publica ejemplos de dashboards que hayas creado para atraer la atención de posibles empleadores.
3. **Grupos y comunidades**: Únete a grupos de Power BI en redes sociales o foros especializados. Muchas veces, las oportunidades surgen de conexiones dentro de estas comunidades.
4. **Freelancing internacional**: Si te sientes cómodo trabajando con clientes de otros países, plataformas como Upwork y Fiverr son excelentes para encontrar proyectos relacionados con Power BI.
5. **Empresas locales**: Contacta directamente a pequeñas empresas en tu área. Muchas necesitan ayuda con la visualización de datos pero no tienen un equipo interno para hacerlo.

¿Te gustaría que te ayude a crear un perfil profesional o a buscar más recursos? 😊

Trabajo en un Data Center y necesito hacer un checklist de ingreso/egreso/ o movimiento interno entre rack de equipos. En primera instancia estaba realizando un Microsoft Form que envie la informacion y llegue a un excel. Ahora podre realizar un codigo en python y crear una app y trabajar con SQL dependiendo de la informacion que ingrese el solicitante de equipos?.

¡Claro que sí, puedes crear una solución más robusta usando Python, una aplicación personalizada y SQL para manejar el flujo de ingreso, egreso o movimiento interno de equipos en tu Data Center! Aquí tienes una idea general de cómo podrías estructurar tu proyecto:

**1. Frontend: La Interfaz de Usuario**

* Puedes usar **Streamlit** o **Tkinter** en Python para crear una aplicación amigable donde los usuarios puedan llenar el checklist.
* La interfaz debería incluir formularios con campos para los datos clave (nombre del solicitante, tipo de movimiento, número de rack, etc.).

**2. Backend: Procesamiento de Datos**

* Cuando el usuario llene el formulario, tu código en Python puede procesar los datos ingresados.
* Librerías como **pandas** pueden ayudarte a validar y dar formato a los datos antes de enviarlos a la base de datos.

**3. Base de Datos: SQL para Almacenamiento**

* Diseña una base de datos en SQL con tablas para almacenar información de ingresos, egresos y movimientos internos.
* Usa una librería como **SQLAlchemy** o **sqlite3** en Python para interactuar con tu base de datos.
* Por ejemplo, puedes insertar nuevos registros cada vez que un usuario complete el checklist.

**4. Automatización de Reportes**

* Si necesitas generar reportes (por ejemplo, en Excel), puedes usar **openpyxl** o **xlsxwriter** en Python para exportar los datos almacenados en la base de datos.
* También podrías crear un dashboard con Power BI que se conecte a tu base de datos para visualizar datos en tiempo real.

**5. Opcional: API REST**

* Si deseas que la app pueda crecer en el futuro, puedes desarrollar una API REST con **Flask** o **FastAPI**. Esto permitiría que otros sistemas se conecten a tu aplicación para interactuar con los datos.

**Flujo Básico de Trabajo**

1. El usuario abre la aplicación y completa el checklist.
2. La información se valida y se envía al backend en Python.
3. Los datos se almacenan en la base de datos SQL.
4. Puedes consultar, filtrar o exportar los datos según sea necesario.

Esto llevaría tu proyecto al siguiente nivel, haciéndolo más eficiente y escalable. ¿Quieres que te ayude con algún detalle técnico o te comparta ejemplos de código para empezar? 🚀

aparte de los pasos que me indicas, una vez que llegue la solicitud al equipo de data center, nosotros debemos enviar la respuesta de la solicitud, por ejemplo, si existe o no existe energia, espacio al solicitante. una vez que esto se concrete y se realice la actividad, deberiamos agregar los datos faltantes al formulario como donde se instalo, rack y ubicacion, ademas de agregar la guia de despacho correspondiente. y este que lleggue a otra base de datos con los datos que se ingreso el solicitante, el dato de respuesta y los datos al finalizar la actividad. cuando este completa la informacion, enviar estos datos a la tabla inventario automaticamente.

¡Entendido, Roberto! Eso suena como un flujo de trabajo muy organizado y funcional para manejar las solicitudes en tu Data Center. La integración de respuestas y actualizaciones automáticas es perfectamente posible utilizando Python, bases de datos SQL y una lógica bien estructurada. Aquí te explico cómo podrías implementar este flujo:

**1. Recepción de la solicitud**

* Los solicitantes llenan el formulario en la aplicación (puedes usar **Streamlit**, como te mencioné antes).
* Los datos ingresados se almacenan en una base de datos inicial, que llamaremos **Base de Datos de Solicitudes**.

**2. Respuesta del equipo del Data Center**

* Tu aplicación podría incluir una sección donde el equipo del Data Center responda la solicitud, especificando si hay energía, espacio, etc.
* Esa respuesta se guarda en la misma base de datos, asociada al ID de la solicitud.

**3. Actualización de datos tras la actividad**

* Una vez realizada la actividad, puedes agregar detalles como el rack, la ubicación y la guía de despacho.
* Tu aplicación puede permitir que el equipo edite la solicitud ya existente para agregar esos datos faltantes.
* Los datos actualizados se pueden enviar automáticamente a una **Base de Datos Finalizada**.

**4. Integración con la tabla de inventario**

* Usa Python para automatizar el envío de los datos completos (los ingresados por el solicitante, la respuesta del equipo y los datos finales) hacia una **Tabla de Inventario** en tu base de datos SQL.
* Esto se puede lograr con triggers en SQL o scripts automáticos programados en Python, que revisen las solicitudes completadas y actualicen la tabla de inventario.

**5. Reportes y gestión**

* Una vez que toda la información está integrada en la Tabla de Inventario, puedes generar reportes o dashboards utilizando Power BI, conectándolo directamente a tu base de datos SQL.
* Esto te permitirá visualizar el flujo de trabajo completo y analizar el rendimiento.

**Librerías útiles:**

* **SQLAlchemy**: Para interactuar con bases de datos SQL desde Python.
* **pandas**: Para procesar y transformar los datos antes de guardarlos en la base de datos.
* **Flask** o **FastAPI**: Si decides crear una API para manejar estas solicitudes.
* **Streamlit**: Para una interfaz de usuario fácil de implementar.

Este flujo asegura que cada etapa esté conectada y automatizada, reduciendo errores y facilitando el trabajo de tu equipo. ¿Quieres que te ayude a diseñar la estructura de las bases de datos o escribir un fragmento de código para uno de estos pasos? 🚀

podemos empezar paso a paso a realizar este proyecto de checklist de data center automatizado porfavor, y detalladamente para no perder ninguna informacion. debo utilizar visual studio code?... favor dime todo al detalle para ir generando el proyecto porfavor

¡Perfecto, Roberto! Vamos paso a paso para realizar tu proyecto de checklist automatizado. Utilizar **Visual Studio Code** es una excelente idea, ya que es un entorno de desarrollo robusto y flexible para trabajar con Python y bases de datos.

**Paso 1: Preparación del entorno**

1. **Instalar Visual Studio Code:**
   * Si no tienes instalado Visual Studio Code, puedes descargarlo desde su página oficial. Instálalo y asegúrate de seleccionar la opción para agregar Python al PATH durante la instalación.
2. **Instalar Python:**
   * Descarga e instala Python desde python.org. Asegúrate de instalar la última versión estable y marca la opción de agregar Python al PATH durante la instalación.
3. **Extensiones necesarias en Visual Studio Code:**
   * Instala las extensiones de:
     + **Python** (Microsoft): Para habilitar el soporte de Python.
     + **SQLite** o cualquier extensión de SQL para interactuar con la base de datos.
     + **Streamlit** (si decides usarlo para el frontend).
4. **Crear un entorno virtual (opcional pero recomendado):**
   * En Visual Studio Code, abre la terminal integrada y ejecuta:

bash

python -m venv venv

* + Activa el entorno virtual:
    - En Windows: .\venv\Scripts\activate
    - En Mac/Linux: source venv/bin/activate

1. **Instalar librerías necesarias:** En la terminal integrada, instala las librerías básicas con:

bash

pip install pandas sqlalchemy streamlit flask openpyxl

**Paso 2: Diseño básico del proyecto**

Vamos a dividir el proyecto en módulos para mantenerlo organizado:

1. **Checklist inicial**: Formulario donde el solicitante ingresa la información.
2. **Respuesta del equipo de Data Center**: Se registra la confirmación sobre energía y espacio.
3. **Actualización tras actividad**: Agregas rack, ubicación y guía de despacho.
4. **Integración con inventario**: Los datos completos se transfieren a la tabla de inventario.

Crea una carpeta principal del proyecto, por ejemplo: ChecklistDataCenter. Dentro de esta carpeta, organiza tus archivos así:

ChecklistDataCenter/

├── app.py # Código principal

├── database.py # Gestión de base de datos

├── templates/ # Archivos HTML (si usas Flask)

├── static/ # Archivos CSS/JS (si usas Flask)

└── utils.py # Funciones auxiliares

**Paso 3: Creación del formulario inicial**

Vamos a empezar con el módulo donde el solicitante llena el formulario inicial. Ejemplo básico en Python usando Streamlit:

**Código:**

python

import streamlit as st

import sqlite3

# Crear conexión a la base de datos

conn = sqlite3.connect("requests.db")

cursor = conn.cursor()

# Crear tabla si no existe

cursor.execute("""

CREATE TABLE IF NOT EXISTS requests (

id INTEGER PRIMARY KEY AUTOINCREMENT,

solicitante TEXT,

tipo\_movimiento TEXT,

rack TEXT,

fecha TEXT

)

""")

conn.commit()

# Formulario inicial

st.title("Checklist de Solicitudes - Data Center")

solicitante = st.text\_input("Nombre del solicitante")

tipo\_movimiento = st.selectbox("Tipo de movimiento", ["Ingreso", "Egreso", "Movimiento interno"])

rack = st.text\_input("Rack")

fecha = st.date\_input("Fecha")

if st.button("Enviar Solicitud"):

cursor.execute("INSERT INTO requests (solicitante, tipo\_movimiento, rack, fecha) VALUES (?, ?, ?, ?)",

(solicitante, tipo\_movimiento, rack, fecha))

conn.commit()

st.success("¡Solicitud enviada correctamente!")

**Paso 4: Respuesta del equipo de Data Center**

En este paso, vamos a agregar una funcionalidad donde el equipo puede responder si hay energía y espacio disponible.

**Código (continuación del anterior):**

python

# Responder solicitudes

st.header("Responder Solicitudes")

solicitudes = cursor.execute("SELECT \* FROM requests").fetchall()

solicitudes\_ids = [solicitud[0] for solicitud in solicitudes]

if solicitudes\_ids:

solicitud\_id = st.selectbox("Seleccione la solicitud a responder", solicitudes\_ids)

energia = st.radio("¿Hay energía disponible?", ["Sí", "No"])

espacio = st.radio("¿Hay espacio disponible?", ["Sí", "No"])

if st.button("Enviar Respuesta"):

cursor.execute("""

UPDATE requests SET energia=?, espacio=? WHERE id=?

""", (energia, espacio, solicitud\_id))

conn.commit()

st.success("¡Respuesta enviada!")

else:

st.warning("No hay solicitudes pendientes.")

**Siguientes pasos**

1. **Agregar funcionalidad para actualizar la información tras la actividad.**
2. **Conectar con la tabla de inventario para finalizar el proceso.**
3. **Automatizar generación de reportes.**

¿Quieres que continuemos desarrollando el siguiente módulo o necesitas ayuda configurando tu entorno? 😊

podrias detallarme mas sobre el entorno y su preparacion.

¡Por supuesto, Roberto! Preparar el entorno correctamente es fundamental para que tu proyecto funcione sin problemas. Aquí te detallo cada paso y algunos consejos adicionales para que configures todo de manera óptima en **Visual Studio Code**.

**Paso 1: Instalar Visual Studio Code**

Si aún no tienes Visual Studio Code, sigue estos pasos:

1. Descárgalo desde su página oficial: Visual Studio Code.
2. Durante la instalación, selecciona la opción de agregarlo al PATH (esto asegura que puedas usar code en la terminal para abrir proyectos directamente).

**Paso 2: Instalar Python**

1. Descarga la última versión estable de Python desde python.org.
2. Durante la instalación, marca la opción de "Agregar Python al PATH". Esto permite ejecutar Python desde cualquier lugar en la terminal.
3. Una vez instalado, verifica que esté funcionando correctamente:
   * Abre la terminal (o el terminal integrado en Visual Studio Code) y escribe:

bash

python --version

Esto debería mostrarte la versión de Python instalada.

**Paso 3: Configurar Visual Studio Code para Python**

1. **Instalar la extensión de Python**:
   * Abre Visual Studio Code y ve a la sección de extensiones (en el menú lateral, el ícono de cuadrados).
   * Busca **Python** (Microsoft) y haz clic en "Instalar".
2. **Activar el soporte de Python**:
   * Crea una carpeta para tu proyecto, por ejemplo: ChecklistDataCenter.
   * Abre esta carpeta desde Visual Studio Code.
   * Crea un nuevo archivo, por ejemplo: app.py. Si todo está configurado correctamente, Visual Studio Code debería detectar que estás trabajando con Python.

**Paso 4: Crear un entorno virtual**

Un entorno virtual te ayudará a mantener las librerías necesarias para tu proyecto sin afectar tu sistema global.

1. Abre la terminal integrada de Visual Studio Code y ejecuta:

bash

python -m venv venv

Esto creará una carpeta llamada venv que contiene tu entorno virtual.

1. **Activar el entorno virtual**:
   * En Windows:

bash

.\venv\Scripts\activate

* + En Mac/Linux:

bash

source venv/bin/activate

1. Una vez activado, deberías ver el nombre del entorno (venv) al inicio de la línea de la terminal.

**Paso 5: Instalar librerías necesarias**

Instala las librerías que usarás para tu proyecto con el siguiente comando:

bash

pip install pandas sqlalchemy streamlit flask openpyxl

Estas librerías te permitirán:

* **pandas**: Manejar y procesar datos.
* **sqlalchemy**: Interactuar con bases de datos SQL.
* **streamlit**: Crear la interfaz de usuario fácilmente.
* **flask**: Opcional, para crear APIs si decides expandir el proyecto.
* **openpyxl**: Trabajar con Excel para reportes.

**Paso 6: Configurar la base de datos**

Si vas a usar SQLite para empezar, no necesitas instalar nada adicional, ya que viene integrado con Python. Para bases de datos más avanzadas como MySQL o PostgreSQL, tendrás que instalar sus respectivos servidores en tu computadora.

Ejemplo para configurar SQLite:

1. Dentro de tu carpeta de proyecto, crea un archivo vacío llamado database.db (puedes cambiar el nombre según tu preferencia).
2. Usa el código que te compartí anteriormente para crear las tablas en esta base de datos.

**Paso 7: Opcional, instalar Git**

Si planeas manejar versiones del proyecto o colaborar con otros, instala Git desde git-scm.com y configura un repositorio en GitHub para mantener tu código organizado.

**Tips adicionales**

* **Organiza tu código**:
  + Mantén diferentes archivos para cada función (por ejemplo, database.py para todo lo relacionado con la base de datos).
  + Usa comentarios en tu código para documentar cada paso.
* **Prueba tu aplicación frecuentemente**:
  + Corre tu aplicación después de cada cambio importante para asegurarte de que todo funciona correctamente.

¿Te gustaría que profundice más sobre alguno de estos pasos o que empecemos con el primer fragmento de código funcional? 😊

.\venv\Scripts\activate : No se puede cargar el archivo C:\Users\Rober\OneDrive\Escritorio\Checklist\_DataCenter\venv\Scripts\Activate.ps1 porque la ejecución de scripts está deshabilitada en este sistema. Para obtener más información, consulta el tema about\_Execution\_Policies en https:/go.microsoft.com/fwlink/?LinkID=135170. En línea: 1 Carácter: 1 + .\venv\Scripts\activate + ~~~~~~~~~~~~~~~~~~~~~~~ + CategoryInfo : SecurityError: (:) [], PSSecurityException + FullyQualifiedErrorId : UnauthorizedAccess

Este error ocurre porque tu sistema tiene deshabilitada la ejecución de scripts en PowerShell debido a las políticas de seguridad. Para solucionarlo, puedes habilitar la ejecución de scripts siguiendo estos pasos:

**1. Abrir PowerShell como administrador**

* Busca "PowerShell" en el menú de inicio de Windows.
* Haz clic derecho sobre PowerShell y selecciona **"Ejecutar como administrador"**.

**2. Verificar la política de ejecución actual**

* En la consola de PowerShell, escribe:

powershell

Get-ExecutionPolicy

* Si ves "Restricted", significa que la ejecución de scripts está deshabilitada.

**3. Cambiar la política de ejecución**

* Para permitir la ejecución de scripts, escribe el siguiente comando:

powershell

Set-ExecutionPolicy -Scope CurrentUser Unrestricted

* Si prefieres una política más restrictiva, puedes usar "RemoteSigned":

powershell

Set-ExecutionPolicy -Scope CurrentUser RemoteSigned

* En ambos casos, se te pedirá confirmación. Escribe "Y" y presiona Enter.

**4. Activar el entorno virtual**

* Vuelve a intentar activar el entorno virtual con:

powershell

.\venv\Scripts\activate

**Importante**

Cambiar la política de ejecución puede tener implicaciones de seguridad. Si prefieres mantener tu sistema más seguro, puedes volver a la política original después de activar tu entorno virtual:

powershell

Set-ExecutionPolicy -Scope CurrentUser Restricted

¿Necesitas más ayuda para completar esta configuración o quieres que revisemos el siguiente paso del proyecto? 😊

termine el paso 5, ahora detallame el Paso 6: Configurar la base de datos

¡Excelente, Roberto! Ahora que has completado el paso 5, vamos a detallar el **Paso 6: Configurar la base de datos**, que será el corazón de tu proyecto, donde toda la información se almacenará y organizará. Empezaremos con algo sencillo y escalable utilizando **SQLite**, ya que viene integrado con Python y es perfecto para proyectos pequeños.

**1. Crear la base de datos**

Para este paso, trabajaremos directamente en Python, creando la estructura de la base de datos con tablas que correspondan a los datos que necesitas (solicitudes, respuestas, actividad final e inventario).

**Código para inicializar la base de datos:**

Crea un archivo llamado database.py en tu proyecto y escribe el siguiente código:

python

import sqlite3

# Crear o conectar a la base de datos

conn = sqlite3.connect("checklist\_data\_center.db")

cursor = conn.cursor()

# Crear tabla de solicitudes

cursor.execute("""

CREATE TABLE IF NOT EXISTS solicitudes (

id INTEGER PRIMARY KEY AUTOINCREMENT,

solicitante TEXT NOT NULL,

tipo\_movimiento TEXT NOT NULL,

rack TEXT,

fecha TEXT,

energia TEXT,

espacio TEXT,

respuesta\_fecha TEXT

)

""")

# Crear tabla de actividad finalizada

cursor.execute("""

CREATE TABLE IF NOT EXISTS actividad\_finalizada (

id INTEGER PRIMARY KEY AUTOINCREMENT,

solicitud\_id INTEGER NOT NULL,

rack\_final TEXT,

ubicacion TEXT,

guia\_despacho TEXT,

actividad\_fecha TEXT,

FOREIGN KEY (solicitud\_id) REFERENCES solicitudes (id)

)

""")

# Crear tabla de inventario

cursor.execute("""

CREATE TABLE IF NOT EXISTS inventario (

id INTEGER PRIMARY KEY AUTOINCREMENT,

solicitud\_id INTEGER NOT NULL,

solicitante TEXT NOT NULL,

rack\_final TEXT,

ubicacion TEXT,

guia\_despacho TEXT,

actividad\_fecha TEXT,

FOREIGN KEY (solicitud\_id) REFERENCES actividad\_finalizada (solicitud\_id)

)

""")

# Guardar cambios y cerrar la conexión

conn.commit()

conn.close()

print("Base de datos inicializada correctamente.")

**Explicación de las tablas:**

* **solicitudes**: Almacena los datos iniciales ingresados por el solicitante y las respuestas del equipo (energía y espacio).
* **actividad\_finalizada**: Contiene los datos adicionales después de que se realiza la actividad (rack final, ubicación, guía de despacho).
* **inventario**: Centraliza todos los datos completados para gestionar el inventario.

**2. Validar la conexión**

Antes de avanzar con el siguiente paso, verifica que el archivo checklist\_data\_center.db se haya creado en tu carpeta de proyecto.

**Cómo probar:**

1. Abre la terminal en Visual Studio Code.
2. Ejecuta el archivo con:

bash

python database.py

1. Si el mensaje "Base de datos inicializada correctamente" aparece, entonces la estructura está lista.

**3. Configuración adicional**

Para interactuar con esta base de datos desde otros módulos (como app.py), necesitaremos funciones específicas en database.py. Por ejemplo:

**Función para agregar solicitudes:**

python

def agregar\_solicitud(solicitante, tipo\_movimiento, rack, fecha):

conn = sqlite3.connect("checklist\_data\_center.db")

cursor = conn.cursor()

cursor.execute("""

INSERT INTO solicitudes (solicitante, tipo\_movimiento, rack, fecha)

VALUES (?, ?, ?, ?)

""", (solicitante, tipo\_movimiento, rack, fecha))

conn.commit()

conn.close()

**Función para actualizar respuestas:**

python

def actualizar\_respuesta(solicitud\_id, energia, espacio, respuesta\_fecha):

conn = sqlite3.connect("checklist\_data\_center.db")

cursor = conn.cursor()

cursor.execute("""

UPDATE solicitudes

SET energia = ?, espacio = ?, respuesta\_fecha = ?

WHERE id = ?

""", (energia, espacio, respuesta\_fecha, solicitud\_id))

conn.commit()

conn.close()

**Siguientes pasos**

Ahora que tenemos la base de datos configurada y las funciones básicas, podemos pasar al **Paso 7**, donde empezaremos a conectar esta base de datos con la aplicación. ¿Te gustaría que detalle cómo integrar este módulo con Streamlit para el formulario inicial? 🚀

favor detallame porfavor donde creo el archivo db, favor se mas detallista al explicarme porfavor

¡Entendido, Roberto! Vamos a detallar completamente dónde y cómo crear el archivo de base de datos checklist\_data\_center.db paso a paso. Este archivo es crucial porque almacenará toda la información de tu proyecto.

**1. Decide dónde quieres almacenar la base de datos**

* Por organización, el archivo de la base de datos debe estar dentro de la carpeta principal de tu proyecto.
* Por ejemplo, si tu proyecto se llama Checklist\_DataCenter, la estructura ideal sería algo como esto:

Checklist\_DataCenter/

├── database.py # Código para gestionar la base de datos

├── checklist\_data\_center.db # Archivo de la base de datos que se creará automáticamente

├── app.py # Código principal de tu aplicación

├── venv/ # Entorno virtual

* Al seguir este orden, el archivo checklist\_data\_center.db será fácil de ubicar y tu código podrá acceder a él de manera directa.

**2. Código para crear la base de datos**

Debido a que **SQLite** es una base de datos basada en archivos, este archivo checklist\_data\_center.db se creará automáticamente **en la misma ubicación donde ejecutes tu script Python**. Si estás trabajando en **Visual Studio Code**, los pasos serían los siguientes:

**Paso 2.1: Crear el archivo Python database.py**

1. En la carpeta principal de tu proyecto (Checklist\_DataCenter), crea un nuevo archivo llamado database.py.
   * Si estás usando Visual Studio Code, haz clic derecho en la carpeta del proyecto (en el explorador lateral) y selecciona "Nuevo archivo".
   * Nómbralo como database.py.

**Paso 2.2: Pega este código en database.py**

Este código inicializa la base de datos y crea las tablas:

python

import sqlite3

# Crear o conectar a la base de datos

conn = sqlite3.connect("checklist\_data\_center.db")

cursor = conn.cursor()

# Crear tabla de solicitudes

cursor.execute("""

CREATE TABLE IF NOT EXISTS solicitudes (

id INTEGER PRIMARY KEY AUTOINCREMENT,

solicitante TEXT NOT NULL,

tipo\_movimiento TEXT NOT NULL,

rack TEXT,

fecha TEXT,

energia TEXT,

espacio TEXT,

respuesta\_fecha TEXT

)

""")

# Crear tabla de actividad finalizada

cursor.execute("""

CREATE TABLE IF NOT EXISTS actividad\_finalizada (

id INTEGER PRIMARY KEY AUTOINCREMENT,

solicitud\_id INTEGER NOT NULL,

rack\_final TEXT,

ubicacion TEXT,

guia\_despacho TEXT,

actividad\_fecha TEXT,

FOREIGN KEY (solicitud\_id) REFERENCES solicitudes (id)

)

""")

# Crear tabla de inventario

cursor.execute("""

CREATE TABLE IF NOT EXISTS inventario (

id INTEGER PRIMARY KEY AUTOINCREMENT,

solicitud\_id INTEGER NOT NULL,

solicitante TEXT NOT NULL,

rack\_final TEXT,

ubicacion TEXT,

guia\_despacho TEXT,

actividad\_fecha TEXT,

FOREIGN KEY (solicitud\_id) REFERENCES actividad\_finalizada (solicitud\_id)

)

""")

# Guardar cambios y cerrar la conexión

conn.commit()

conn.close()

print("Base de datos inicializada correctamente.")

**3. Ejecutar el archivo para crear la base de datos**

Ahora que el archivo database.py está listo, vamos a ejecutarlo para crear el archivo checklist\_data\_center.db:

1. **Abre la terminal de Visual Studio Code**:
   * Puedes usar el atajo Ctrl + (comilla invertida) para abrirla.
   * Asegúrate de que la terminal esté ubicada en el directorio de tu proyecto (el mismo que contiene database.py).
2. **Ejecuta el archivo Python**:
   * Escribe el siguiente comando en la terminal:

bash

python database.py

1. **Verifica el resultado**:
   * Si todo está bien, deberías ver el mensaje **"Base de datos inicializada correctamente"** en la terminal.
   * Además, deberías ver un nuevo archivo llamado checklist\_data\_center.db en tu carpeta principal del proyecto.

**4. Revisar el archivo de base de datos**

* Si quieres confirmar que el archivo checklist\_data\_center.db contiene las tablas creadas:
  + Abre el archivo con cualquier herramienta de SQLite, como:
    - **DB Browser for SQLite**: Descargar aquí.
    - **Extensiones de SQL en Visual Studio Code** (puedes buscar "SQLite" en la sección de extensiones y usarla para explorar la base de datos directamente desde VS Code).
  + Inspecciona las tablas y verifica que se hayan creado correctamente (solicitudes, actividad\_finalizada, inventario).

**5. Qué hacer después**

Ahora que el archivo de la base de datos está creado y configurado, podemos proceder al siguiente paso: **Conectar la base de datos a tu aplicación para manejar solicitudes y respuestas**. ¿Quieres que detalle cómo integrar esto en tu aplicación o necesitas más aclaraciones sobre esta configuración? 😊

antes de pega el codigo anterior, te puedo indicar que datos debe venir en el checklist de ingreso, en el checklist de egreso y de movimiento. a su vez de darte esta informacion, tambien en la solicitud en la parte donde el solicitante ponga la marca y modelo de equipo, yo tengo una base de datos en excel llamado catalogo, la idea igual seria que en la solicitud pueda darle al solicitante la opcion de buscar la marca y modelo a traves de este archivo. te puedo dar todos estos datos para que sea mas facil

¡Perfecto, Roberto! Por supuesto, sería muy útil que me proporciones todos los detalles sobre los datos necesarios en el checklist de **ingreso**, **egreso** y **movimiento**. Además, podemos integrar el archivo Excel con el catálogo de marcas y modelos en el sistema para que los solicitantes puedan buscar y seleccionar las opciones de manera interactiva.

A continuación, detallo cómo podríamos estructurar y desarrollar esto de manera óptima:

**Checklist específico por tipo de movimiento**

1. **Checklist de ingreso:**
   * Nombre del solicitante.
   * Fecha del ingreso.
   * Marca y modelo del equipo (con opción de búsqueda en el archivo Excel del catálogo).
   * Número de serie.
   * Tipo de equipo (servidor, switch, etc.).
   * Rack asignado para instalación.
   * Ubicación física en el rack (U-height).
   * Estado inicial (nuevo/usado).
   * Detalles adicionales o notas.
2. **Checklist de egreso:**
   * Nombre del solicitante.
   * Fecha del egreso.
   * Marca y modelo del equipo (con opción de búsqueda en el archivo Excel del catálogo).
   * Número de serie.
   * Rack de origen.
   * Motivo del egreso (desinstalación, traslado, etc.).
   * Estado del equipo al salir (funcional/dañado).
   * Detalles adicionales o notas.
3. **Checklist de movimiento interno:**
   * Nombre del solicitante.
   * Fecha del movimiento.
   * Marca y modelo del equipo (con opción de búsqueda en el archivo Excel del catálogo).
   * Número de serie.
   * Rack de origen.
   * Rack de destino.
   * Nueva ubicación física en el rack (U-height).
   * Motivo del movimiento (reorganización, ampliación, etc.).
   * Detalles adicionales o notas.

**Integración del archivo Excel del catálogo**

Para que el solicitante pueda buscar y seleccionar marca y modelo de equipo, podemos hacer lo siguiente:

1. **Importar el archivo Excel en Python**:
   * Usaremos la librería **pandas** para leer los datos del archivo Excel.
   * Generaremos una lista desplegable con las marcas y modelos disponibles en el catálogo.
2. **Búsqueda interactiva**:
   * El sistema puede permitir que el solicitante filtre por marca o modelo de equipo, proporcionando resultados basados en el archivo Excel.

**Ejemplo de código:**

python

import pandas as pd

# Cargar el catálogo desde el archivo Excel

catalogo\_df = pd.read\_excel("catalogo.xlsx")

# Función para buscar marcas y modelos

def obtener\_marcas\_modelos():

# Combinar columna de marca y modelo para lista desplegable

catalogo\_df["marca\_modelo"] = catalogo\_df["marca"] + " - " + catalogo\_df["modelo"]

return catalogo\_df["marca\_modelo"].tolist()

# Mostrar opciones al solicitante (ejemplo con Streamlit)

import streamlit as st

marca\_modelo\_seleccionado = st.selectbox("Seleccione marca y modelo", obtener\_marcas\_modelos())

st.write(f"Has seleccionado: {marca\_modelo\_seleccionado}")

**Pasos siguientes**

* Si me proporcionas los datos exactos que deben ir en cada checklist, puedo ajustar el código para incluirlos.
* También puedo configurar el sistema para cargar el catálogo directamente desde tu archivo Excel. Si necesitas, puedes compartir el formato de las columnas del archivo (por ejemplo, "marca", "modelo", "tipo de equipo", etc.).

Con esta información, podríamos personalizar completamente el proyecto y avanzar con una solución detallada. ¿Qué datos quieres incluir y cómo está estructurado tu archivo Excel del catálogo? 😊

Checklist de ingreso: Nombre del solicitante. Fecha del ingreso. Cliente (tengo un listado en excel con los clientes) DataCenter (Providencia/San Bernardo/Ascentys/Bodega) Ticket (Si existe un ticket agregarlo, sino, poner pendiente o que de una opcion al usuario) Cuantos equipos ingresara? (que de la opcion de cantidad) Marca y modelo del equipo (con opción de búsqueda en el archivo Excel del catálogo). Número de serie. Tipo de equipo (servidor, switch, etc.). (Este dato viene en el catalogo, si el usuario elige la marca y modelo, este dato se deberia llenar automaticamente desde el excel catalogo) los datos anteriores debes ser obligatorios para el solicitante estos datos siguientes deberian ser para el team de Data Center al momento de la instalacion o guardado en bodega. Rack asignado para instalación. Ubicación física en el rack (U-height). Cantidad de U utilizando en el rack Propiedad Sala Dual/Single Tag Mt2 Estado inicial (nuevo/usado). Detalles adicionales o notas. Checklist de egreso: Nombre del solicitante. Fecha del egreso. Marca y modelo del equipo (con opción de búsqueda en el archivo Excel del catálogo). Número de serie. Rack de origen. Motivo del egreso (desinstalación, traslado, etc.). Estado del equipo al salir (funcional/dañado). Detalles adicionales o notas. Checklist de movimiento interno: Nombre del solicitante. Fecha del movimiento. Marca y modelo del equipo (con opción de búsqueda en el archivo Excel del catálogo). Número de serie. Rack de origen. Rack de destino. Nueva ubicación física en el rack (U-height). Motivo del movimiento (reorganización, ampliación, etc.). Detalles adicionales o notas.

¡Perfecto! Esto me da una visión muy completa y permitirá que organicemos todo de forma eficiente. Empecemos a estructurar el proyecto teniendo en cuenta los datos específicos de los checklists.

**Paso 6 (extendido): Configurar la base de datos**

Con base en los datos que me proporcionaste, actualizaremos el diseño de las tablas en la base de datos para incluir todos los campos necesarios.

**Diseño de las tablas en la base de datos:**

1. **Tabla** solicitudes\_ingreso:
   * Contendrá los datos iniciales proporcionados por el solicitante al momento del ingreso.
   * Campos:
     + id (llave primaria).
     + solicitante.
     + fecha\_ingreso.
     + cliente.
     + datacenter.
     + ticket.
     + cantidad\_equipos.
     + marca\_modelo.
     + numero\_serie.
     + tipo\_equipo (autocompletado desde el catálogo).
   * Datos del equipo de Data Center:
     + rack\_asignado.
     + ubicacion\_rack\_u.
     + cantidad\_u.
     + propiedad.
     + sala.
     + dual\_single.
     + tag.
     + mt2.
     + estado\_inicial.
     + detalles.
2. **Tabla** solicitudes\_egreso:
   * Contendrá los datos del checklist de egreso.
   * Campos:
     + id (llave primaria).
     + solicitante.
     + fecha\_egreso.
     + marca\_modelo.
     + numero\_serie.
     + rack\_origen.
     + motivo\_egreso.
     + estado\_salida.
     + detalles.
3. **Tabla** movimientos\_internos:
   * Almacena los datos del checklist de movimiento interno.
   * Campos:
     + id (llave primaria).
     + solicitante.
     + fecha\_movimiento.
     + marca\_modelo.
     + numero\_serie.
     + rack\_origen.
     + rack\_destino.
     + ubicacion\_rack\_u.
     + motivo\_movimiento.
     + detalles.
4. **Tabla** catalogo (para cargar el Excel):
   * Almacena las marcas, modelos y tipos de equipos.
   * Campos:
     + id (llave primaria).
     + marca.
     + modelo.
     + tipo\_equipo.

**Código para configurar estas tablas en la base de datos:**

Modificaremos el archivo database.py para que incluya las nuevas tablas:

python

import sqlite3

# Crear o conectar a la base de datos

conn = sqlite3.connect("checklist\_data\_center.db")

cursor = conn.cursor()

# Crear tabla de ingreso

cursor.execute("""

CREATE TABLE IF NOT EXISTS solicitudes\_ingreso (

id INTEGER PRIMARY KEY AUTOINCREMENT,

solicitante TEXT NOT NULL,

fecha\_ingreso TEXT NOT NULL,

cliente TEXT NOT NULL,

datacenter TEXT NOT NULL,

ticket TEXT NOT NULL,

cantidad\_equipos INTEGER NOT NULL,

marca\_modelo TEXT NOT NULL,

numero\_serie TEXT NOT NULL,

tipo\_equipo TEXT NOT NULL,

rack\_asignado TEXT,

ubicacion\_rack\_u TEXT,

cantidad\_u INTEGER,

propiedad TEXT,

sala TEXT,

dual\_single TEXT,

tag TEXT,

mt2 TEXT,

estado\_inicial TEXT,

detalles TEXT

)

""")

# Crear tabla de egreso

cursor.execute("""

CREATE TABLE IF NOT EXISTS solicitudes\_egreso (

id INTEGER PRIMARY KEY AUTOINCREMENT,

solicitante TEXT NOT NULL,

fecha\_egreso TEXT NOT NULL,

marca\_modelo TEXT NOT NULL,

numero\_serie TEXT NOT NULL,

rack\_origen TEXT NOT NULL,

motivo\_egreso TEXT NOT NULL,

estado\_salida TEXT NOT NULL,

detalles TEXT

)

""")

# Crear tabla de movimiento interno

cursor.execute("""

CREATE TABLE IF NOT EXISTS movimientos\_internos (

id INTEGER PRIMARY KEY AUTOINCREMENT,

solicitante TEXT NOT NULL,

fecha\_movimiento TEXT NOT NULL,

marca\_modelo TEXT NOT NULL,

numero\_serie TEXT NOT NULL,

rack\_origen TEXT NOT NULL,

rack\_destino TEXT NOT NULL,

ubicacion\_rack\_u TEXT NOT NULL,

motivo\_movimiento TEXT NOT NULL,

detalles TEXT

)

""")

# Crear tabla del catálogo

cursor.execute("""

CREATE TABLE IF NOT EXISTS catalogo (

id INTEGER PRIMARY KEY AUTOINCREMENT,

marca TEXT NOT NULL,

modelo TEXT NOT NULL,

tipo\_equipo TEXT NOT NULL

)

""")

# Guardar cambios y cerrar conexión

conn.commit()

conn.close()

print("Base de datos configurada correctamente.")

**Integración del Excel del catálogo**

Ahora, cargaremos los datos del archivo Excel del catálogo en la tabla catalogo. Puedes agregar la funcionalidad en el mismo archivo database.py o en un script separado para cargar los datos.

**Código para cargar el catálogo desde Excel:**

Guarda tu archivo Excel como catalogo.xlsx en la carpeta principal del proyecto. Luego, utiliza este script para cargarlo en la base de datos:

python

import pandas as pd

import sqlite3

# Leer el archivo Excel

catalogo\_df = pd.read\_excel("catalogo.xlsx")

# Conectar a la base de datos

conn = sqlite3.connect("checklist\_data\_center.db")

cursor = conn.cursor()

# Insertar datos en la tabla catalogo

for \_, row in catalogo\_df.iterrows():

cursor.execute("""

INSERT INTO catalogo (marca, modelo, tipo\_equipo)

VALUES (?, ?, ?)

""", (row["marca"], row["modelo"], row["tipo\_equipo"]))

conn.commit()

conn.close()

print("Datos del catálogo cargados correctamente.")

**Siguientes pasos**

1. Validar que todas las tablas se creen correctamente ejecutando database.py.
2. Probar la carga de datos del archivo Excel.
3. Continuar con la integración en la aplicación, permitiendo que los usuarios seleccionen datos del catálogo y llenen los checklists.

¿Te gustaría que avancemos con la interfaz para el checklist de ingreso o necesitas más detalles sobre algún paso aquí? 😊

Traceback (most recent call last): File "c:\Users\Rober\OneDrive\Escritorio\Checklist\_DataCenter\carga\_data.py", line 13, in <module> cursor.execute(""" ~~~~~~~~~~~~~~^^^^ INSERT INTO catalogo (marca, modelo, tipo\_equipo) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ VALUES (?, ?, ?) ^^^^^^^^^^^^^^^^ """, (row["marca"], row["modelo"], row["tipo\_equipo"])) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ sqlite3.IntegrityError: NOT NULL constraint failed: catalogo.modelo

Este error ocurre porque el campo modelo de la tabla catalogo tiene una restricción **NOT NULL**, lo que significa que no puede estar vacío. Sin embargo, al intentar insertar datos desde el archivo Excel, parece que hay filas donde la columna **modelo** está vacía o tiene valores nulos.

Vamos a corregirlo paso a paso:

**1. Verificar el archivo Excel**

Abre el archivo catalogo.xlsx y revisa si hay filas donde la columna **modelo** está vacía. Esto podría ocurrir debido a:

* Errores en el archivo Excel (datos faltantes o incompletos).
* Formato incorrecto de la columna en Excel.

**2. Solución en Python**

Para evitar insertar valores nulos en la base de datos, puedes filtrar las filas del archivo Excel en tu script antes de insertarlas. Aquí tienes el código modificado:

**Código actualizado para insertar datos:**

python

import pandas as pd

import sqlite3

# Leer el archivo Excel

catalogo\_df = pd.read\_excel("catalogo.xlsx")

# Filtrar filas donde modelo no esté vacío

catalogo\_df = catalogo\_df.dropna(subset=["modelo"])

# Conectar a la base de datos

conn = sqlite3.connect("checklist\_data\_center.db")

cursor = conn.cursor()

# Insertar datos en la tabla catalogo

for \_, row in catalogo\_df.iterrows():

cursor.execute("""

INSERT INTO catalogo (marca, modelo, tipo\_equipo)

VALUES (?, ?, ?)

""", (row["marca"], row["modelo"], row["tipo\_equipo"]))

conn.commit()

conn.close()

print("Datos del catálogo cargados correctamente.")

**3. ¿Qué hace este código?**

* catalogo\_df.dropna(subset=["modelo"]): Esta línea elimina cualquier fila donde la columna modelo tenga valores vacíos o nulos.
* El resto del script sigue igual, pero ahora solo inserta datos válidos en la base de datos.

**4. Validar la carga**

Ejecuta el archivo carga\_data.py nuevamente después de aplicar estos cambios. Si el error persiste, verifica lo siguiente:

* Asegúrate de que la tabla catalogo en la base de datos tiene las columnas correctamente configuradas (marca, modelo, tipo\_equipo).
* Confirma que la estructura del archivo Excel coincide con las columnas esperadas.

Si aún necesitas ayuda, puedo guiarte para revisar el contenido del archivo Excel o depurar tu código con más detalle. ¡Avísame cómo te va! 🚀

Traceback (most recent call last): File "c:\Users\Rober\OneDrive\Escritorio\Checklist\_DataCenter\carga\_data.py", line 16, in <module> cursor.execute(""" ~~~~~~~~~~~~~~^^^^ INSERT INTO catalogo (marca, modelo, tipo\_equipo) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ VALUES (?, ?, ?) ^^^^^^^^^^^^^^^^ """, (row["marca"], row["modelo"], row["tipo\_equipo"])) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ sqlite3.IntegrityError: NOT NULL constraint failed: catalogo.marca

Este error ocurre porque el campo marca en la tabla catalogo tiene una restricción **NOT NULL** y alguna de las filas en tu archivo Excel tiene la columna **marca** vacía o nula. Veamos cómo solucionarlo paso a paso:

**1. Revisar el archivo Excel**

Abre tu archivo catalogo.xlsx y verifica si hay filas donde la columna **marca** está vacía o contiene valores nulos. Esto puede suceder por:

* Errores de datos en el archivo (información faltante).
* Celdas en blanco en la columna marca.

Si encuentras celdas vacías, completa esas filas manualmente con los datos faltantes o elimínalas si no son necesarias.

**2. Filtrar filas nulas desde el código**

Si prefieres manejar esto directamente en tu código, puedes filtrar las filas con datos nulos antes de insertarlas en la base de datos.

**Código actualizado para evitar filas con marca vacía:**

python

import pandas as pd

import sqlite3

# Leer el archivo Excel

catalogo\_df = pd.read\_excel("catalogo.xlsx")

# Filtrar filas donde marca o modelo no estén vacíos

catalogo\_df = catalogo\_df.dropna(subset=["marca", "modelo"])

# Conectar a la base de datos

conn = sqlite3.connect("checklist\_data\_center.db")

cursor = conn.cursor()

# Insertar datos en la tabla catalogo

for \_, row in catalogo\_df.iterrows():

cursor.execute("""

INSERT INTO catalogo (marca, modelo, tipo\_equipo)

VALUES (?, ?, ?)

""", (row["marca"], row["modelo"], row["tipo\_equipo"]))

conn.commit()

conn.close()

print("Datos del catálogo cargados correctamente.")

**¿Qué hace este cambio?**

1. catalogo\_df.dropna(subset=["marca", "modelo"]):
   * Elimina cualquier fila del archivo Excel donde las columnas marca o modelo estén vacías.
   * Esto asegura que los datos incompletos no lleguen a la base de datos.

**3. Validar las filas antes de insertar**

Para asegurarte de que cada fila tiene datos válidos antes de insertarla, puedes incluir una validación explícita en el bucle de inserción:

**Código para validar datos fila por fila:**

python

for \_, row in catalogo\_df.iterrows():

if pd.notna(row["marca"]) and pd.notna(row["modelo"]):

cursor.execute("""

INSERT INTO catalogo (marca, modelo, tipo\_equipo)

VALUES (?, ?, ?)

""", (row["marca"], row["modelo"], row["tipo\_equipo"]))

Esto asegura que solo se inserten filas donde marca y modelo tengan valores válidos.

**4. Probar nuevamente**

Una vez realizados los cambios, guarda el script y ejecútalo nuevamente con:

bash

python carga\_data.py

Si el archivo Excel contiene datos válidos, este código debería cargar el catálogo correctamente en la base de datos.

**¿Qué hacer si el problema persiste?**

Si el error persiste, revisemos:

1. **Estructura del archivo Excel**:
   * Asegúrate de que las columnas **marca**, **modelo** y **tipo\_equipo** están correctamente escritas y alineadas con lo que espera el script.
2. **Tabla de la base de datos**:
   * Verifica que la tabla catalogo tenga las columnas correctamente configuradas y que todas acepten datos.

Estoy aquí para ayudarte con cualquier ajuste adicional. ¿Cómo te fue con estas soluciones? 😊

Traceback (most recent call last): File "c:\Users\Rober\OneDrive\Escritorio\Checklist\_DataCenter\carga\_data.py", line 28, in <module> cursor.execute(""" ~~~~~~~~~~~~~~^^^^ INSERT INTO catalogo (marca, modelo, tipo\_equipo) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ VALUES (?, ?, ?) ^^^^^^^^^^^^^^^^ """, (row["marca"], row["modelo"], row["tipo\_equipo"])) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ sqlite3.ProgrammingError: Cannot operate on a closed database.

Este error ocurre porque el objeto de conexión a la base de datos (conn) se cerró antes de que se intentara ejecutar el cursor.execute() en tu código. Una vez que se llama a conn.close(), el cursor también se considera no operativo, lo que provoca este error.

Vamos a corregir esto asegurándonos de que la conexión a la base de datos permanezca abierta mientras se insertan los datos del catálogo. Aquí está la versión ajustada del código:

**Código corregido: Mantener abierta la conexión**

python

import pandas as pd

import sqlite3

# Leer el archivo Excel

catalogo\_df = pd.read\_excel("catalogo.xlsx")

# Filtrar filas donde marca o modelo no estén vacíos

catalogo\_df = catalogo\_df.dropna(subset=["marca", "modelo"])

# Conectar a la base de datos

conn = sqlite3.connect("checklist\_data\_center.db")

cursor = conn.cursor()

try:

# Insertar datos en la tabla catalogo

for \_, row in catalogo\_df.iterrows():

cursor.execute("""

INSERT INTO catalogo (marca, modelo, tipo\_equipo)

VALUES (?, ?, ?)

""", (row["marca"], row["modelo"], row["tipo\_equipo"]))

conn.commit() # Guardar los cambios después de insertar los datos

print("Datos del catálogo cargados correctamente.")

except Exception as e:

print(f"Error al cargar los datos: {e}")

finally:

# Cerrar la conexión en el bloque finally para garantizar que no quede abierta

conn.close()

**Qué cambia en este código:**

1. **Uso de** try **y** finally**:**
   * Usamos un bloque try para asegurarnos de capturar cualquier error durante la ejecución del código.
   * El bloque finally garantiza que la conexión a la base de datos siempre se cerrará correctamente, sin importar si hubo un error o no.
2. **Conexión abierta hasta que se termine la inserción:**
   * La conexión a la base de datos permanece activa (conn no se cierra) hasta que se han insertado todos los datos y se ejecuta conn.commit() para guardar los cambios.
3. **Manejo de errores:**
   * Si ocurre un error durante la carga, este será capturado y se imprimirá un mensaje en la terminal, ayudando a depurar el problema.

**Pasos para ejecutar**

1. Guarda este código en el archivo carga\_data.py.
2. Ejecuta el archivo desde la terminal:

bash

python carga\_data.py

1. Deberías ver el mensaje **"Datos del catálogo cargados correctamente."** si todo funciona correctamente.

**Validación adicional**

Para verificar que los datos se cargaron en la tabla catalogo:

1. Abre la base de datos con un visualizador SQLite, como:
   * **DB Browser for SQLite**: Descargar aquí.
2. Inspecciona la tabla catalogo y verifica que los datos del Excel se hayan insertado correctamente.

Si encuentras algún otro error o necesitas más ayuda, ¡aquí estoy! 🚀

import pandas as pd import sqlite3 # Leer el archivo Excel catalogo\_df = pd.read\_excel("Catalogo\_full.xlsx") # Filtrar filas donde modelo no esté vacío catalogo\_df = catalogo\_df.dropna(subset=["marca", "modelo", "tipo\_equipo"]) # Conectar a la base de datos conn = sqlite3.connect("checklist\_data\_center.db") cursor = conn.cursor() # Insertar datos en la tabla catalogo for \_, row in catalogo\_df.iterrows(): cursor.execute(""" INSERT INTO catalogo (marca, modelo, tipo\_equipo) VALUES (?, ?, ?) """, (row["marca"], row["modelo"], row["tipo\_equipo"])) conn.commit() conn.close() print("Datos del catálogo cargados correctamente.") for \_, row in catalogo\_df.iterrows(): if pd.notna(row["marca"]) and pd.notna(row["modelo"]): cursor.execute(""" INSERT INTO catalogo (marca, modelo, tipo\_equipo) VALUES (?, ?, ?) """, (row["marca"], row["modelo"], row["tipo\_equipo"])) y aparece este error Traceback (most recent call last): File "c:\Users\Rober\OneDrive\Escritorio\Checklist\_DataCenter\carga\_data.py", line 28, in <module> cursor.execute(""" ~~~~~~~~~~~~~~^^^^ INSERT INTO catalogo (marca, modelo, tipo\_equipo) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ VALUES (?, ?, ?) ^^^^^^^^^^^^^^^^ """, (row["marca"], row["modelo"], row["tipo\_equipo"])) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ sqlite3.ProgrammingError: Cannot operate on a closed database.

El error "Cannot operate on a closed database" ocurre porque estás intentando usar cursor.execute() **después** de haber cerrado la conexión a la base de datos con conn.close(). Esto sucede en tu código porque tienes dos bucles: el primero funciona bien y luego cierras la conexión. Pero el segundo bucle intenta acceder a la base de datos usando el cursor, lo que ya no es posible.

Aquí tienes cómo corregirlo:

**Solución completa: Eliminar el segundo bucle**

El segundo bucle no es necesario, ya que el primer bucle ya inserta todos los datos en la base de datos. Por lo tanto, el segundo bucle debe ser eliminado. Aquí está el código corregido:

**Código limpio y funcional:**

python

import pandas as pd

import sqlite3

# Leer el archivo Excel

catalogo\_df = pd.read\_excel("Catalogo\_full.xlsx")

# Filtrar filas donde marca, modelo y tipo\_equipo no estén vacíos

catalogo\_df = catalogo\_df.dropna(subset=["marca", "modelo", "tipo\_equipo"])

# Conectar a la base de datos

conn = sqlite3.connect("checklist\_data\_center.db")

cursor = conn.cursor()

# Insertar datos en la tabla catalogo

for \_, row in catalogo\_df.iterrows():

cursor.execute("""

INSERT INTO catalogo (marca, modelo, tipo\_equipo)

VALUES (?, ?, ?)

""", (row["marca"], row["modelo"], row["tipo\_equipo"]))

# Guardar cambios y cerrar conexión

conn.commit()

conn.close()

print("Datos del catálogo cargados correctamente.")

**¿Qué cambió en este código?**

1. **Eliminado el segundo bucle:**
   * El segundo bucle duplicaba el intento de insertar los mismos datos, lo cual era redundante y no necesario.
   * Toda la lógica para insertar los datos ya está correctamente implementada en el primer bucle.
2. **Cierre de la conexión después de que todo el trabajo se haya completado:**
   * conn.close() ahora ocurre una sola vez al final del programa, asegurándose de que el cursor y la conexión estén disponibles durante todo el proceso de inserción.

**Pasos para ejecutar**

1. Guarda este código en tu archivo carga\_data.py.
2. Ejecuta el archivo desde la terminal:

bash

python carga\_data.py

**Validar que los datos se cargaron correctamente**

1. Abre la base de datos checklist\_data\_center.db con una herramienta como **DB Browser for SQLite** o cualquier extensión de SQLite en Visual Studio Code.
2. Verifica que los datos en la tabla catalogo coincidan con los del archivo Excel Catalogo\_full.xlsx.

**Consideraciones adicionales**

* Si sigues teniendo problemas, verifica el contenido de tu archivo Excel:
  + Asegúrate de que las columnas marca, modelo y tipo\_equipo tienen valores en todas las filas importantes.
* Si es necesario, comparte detalles del archivo Excel (como ejemplos de las filas) para validar que el procesamiento sea correcto.

Con estos ajustes, tu script debería cargar los datos sin errores. ¿Cómo te fue? 😊

perfecto continuemos

¡Excelente, Roberto! Ahora que tenemos el catálogo cargado en la base de datos, podemos proceder a implementar la **interfaz inicial del checklist de ingreso**, donde el solicitante pueda llenar los datos requeridos y seleccionar marca y modelo desde el catálogo.

**Paso 7: Crear la interfaz para el checklist de ingreso**

Utilizaremos **Streamlit** para desarrollar una interfaz gráfica sencilla e interactiva. El checklist permitirá a los usuarios:

1. Llenar datos obligatorios como solicitante, fecha, cliente, DataCenter, etc.
2. Seleccionar la marca y modelo del equipo desde el catálogo cargado en la base de datos.
3. Guardar los datos en la tabla solicitudes\_ingreso.

**Archivo: app.py**

Crea un archivo llamado app.py y copia este código base:

python

import streamlit as st

import sqlite3

import pandas as pd

# Función para conectar a la base de datos

def conectar\_db():

conn = sqlite3.connect("checklist\_data\_center.db")

return conn

# Función para obtener clientes desde el catálogo

def obtener\_clientes():

# Aquí puedes conectar otra tabla o cargar desde Excel si es necesario.

return ["Cliente A", "Cliente B", "Cliente C"]

# Función para obtener marcas y modelos desde el catálogo

def obtener\_catalogo():

conn = conectar\_db()

catalogo\_df = pd.read\_sql\_query("SELECT marca, modelo, tipo\_equipo FROM catalogo", conn)

conn.close()

catalogo\_df["marca\_modelo"] = catalogo\_df["marca"] + " - " + catalogo\_df["modelo"]

return catalogo\_df

# Crear la interfaz

st.title("Checklist de Ingreso - Data Center")

# Formulario inicial

st.subheader("Datos iniciales del solicitante")

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_ingreso = st.date\_input("Fecha de ingreso:")

cliente = st.selectbox("Seleccione el cliente:", obtener\_clientes())

datacenter = st.selectbox("Seleccione el DataCenter:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

ticket = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):")

cantidad\_equipos = st.number\_input("Cantidad de equipos a ingresar:", min\_value=1, step=1)

st.subheader("Datos de los equipos")

catalogo\_df = obtener\_catalogo()

marca\_modelo\_seleccionado = st.selectbox("Seleccione la marca y modelo del equipo:", catalogo\_df["marca\_modelo"])

numero\_serie = st.text\_input("Número de serie del equipo:")

# Buscar tipo de equipo automáticamente desde el catálogo

tipo\_equipo = catalogo\_df[catalogo\_df["marca\_modelo"] == marca\_modelo\_seleccionado]["tipo\_equipo"].values[0]

st.write(f"Tipo de equipo (automático): \*\*{tipo\_equipo}\*\*")

# Botón para guardar

if st.button("Guardar solicitud"):

if all([solicitante, cliente, datacenter, ticket, numero\_serie]):

conn = conectar\_db()

cursor = conn.cursor()

cursor.execute("""

INSERT INTO solicitudes\_ingreso (

solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

marca\_modelo, numero\_serie, tipo\_equipo

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

marca\_modelo\_seleccionado, numero\_serie, tipo\_equipo))

conn.commit()

conn.close()

st.success("¡Solicitud de ingreso guardada correctamente!")

else:

st.error("Por favor, complete todos los campos obligatorios.")

**Explicación del código**

1. **Conexión a la base de datos:**
   * La función conectar\_db() establece una conexión con checklist\_data\_center.db.
   * Esto permite interactuar fácilmente con la base de datos desde cualquier parte del código.
2. **Datos del catálogo:**
   * obtener\_catalogo() carga los datos de la tabla catalogo en un DataFrame de pandas.
   * Se crea una columna combinada marca\_modelo para que el usuario pueda seleccionar marca y modelo juntos.
3. **Formulario interactivo:**
   * Streamlit ofrece entradas como text\_input, date\_input y selectbox para capturar datos del usuario.
   * Los campos obligatorios aseguran que el formulario se complete antes de guardar.
4. **Guardar en la base de datos:**
   * Los datos ingresados se guardan en la tabla solicitudes\_ingreso mediante un comando SQL.
5. **Validación automática:**
   * Si falta algún dato obligatorio, se muestra un mensaje de error.
   * El tipo de equipo se obtiene automáticamente del catálogo usando pandas.

**Pasos para probar**

1. Guarda este código en el archivo app.py.
2. Abre una terminal, activa tu entorno virtual (venv) y ejecuta:

bash

streamlit run app.py

1. Se abrirá una ventana en tu navegador mostrando la interfaz del checklist de ingreso.
2. Completa el formulario y verifica que los datos se guarden en la base de datos.

**Siguientes pasos**

1. Validar que las solicitudes se guarden correctamente en la tabla solicitudes\_ingreso.
2. Implementar funcionalidades adicionales, como el checklist de egreso y movimiento interno.
3. Crear una interfaz para que el equipo del Data Center gestione las solicitudes (como asignar racks y agregar detalles finales).

¿Quieres que avancemos con la validación de datos o prefieres que empecemos con otro checklist? 😊

Pasos para probar Guarda este código en el archivo app.py. Abre una terminal, activa tu entorno virtual (venv) y ejecuta: me ayudas en este paso , mas detalle como hacerlo

¡Claro, Roberto! Aquí está el proceso detallado para ejecutar el archivo app.py con **Streamlit** en tu entorno virtual (venv):

**Paso 1: Verifica que el archivo app.py esté guardado**

1. Asegúrate de que el código que compartí esté correctamente copiado en un archivo llamado app.py.
2. Coloca el archivo en la carpeta principal de tu proyecto (donde también se encuentra tu base de datos y el entorno virtual).

**Paso 2: Abre una terminal**

1. Si estás usando **Visual Studio Code**:
   * Abre tu proyecto en VS Code.
   * Abre la terminal integrada (atajo: `Ctrl + ``, es decir, la tecla de comilla invertida).
2. Si prefieres usar otra terminal, como **PowerShell** o **Command Prompt**, asegúrate de navegar hasta la carpeta de tu proyecto usando el comando:

bash

cd "Ruta\_de\_tu\_carpeta"

Por ejemplo, si tu proyecto está en el escritorio:

bash

cd "C:\Users\Rober\OneDrive\Escritorio\Checklist\_DataCenter"

**Paso 3: Activa tu entorno virtual (venv)**

Para activar tu entorno virtual, sigue los pasos según tu sistema operativo:

**Windows:**

1. En la terminal, ejecuta:

bash

.\venv\Scripts\activate

1. Verás que el nombre del entorno virtual (venv) aparece al inicio de la línea de la terminal, indicando que está activo.

**Mac/Linux:**

1. En la terminal, ejecuta:

bash

source venv/bin/activate

1. Al igual que en Windows, deberías ver venv en la línea de la terminal.

**Paso 4: Instala Streamlit en el entorno virtual**

Si aún no has instalado **Streamlit**, asegúrate de hacerlo con el siguiente comando:

bash

pip install streamlit

**Paso 5: Ejecuta tu archivo app.py**

Con el entorno virtual activado, ejecuta el archivo de la aplicación usando:

bash

streamlit run app.py

Streamlit abrirá automáticamente tu navegador predeterminado y cargará la interfaz en una nueva pestaña.

**Paso 6: Interactúa con la interfaz**

1. Completa los datos en el formulario (solicitante, fecha, cliente, etc.).
2. Guarda la información y verifica que los datos se envían correctamente a la base de datos.

**Solución de problemas**

Si algo no funciona:

1. **Error "No se encuentra Streamlit":**
   * Asegúrate de haber activado correctamente el entorno virtual antes de instalar y ejecutar Streamlit.
2. **No se abre el navegador automáticamente:**
   * Copia y pega el enlace que aparece en la terminal (algo como http://localhost:8501) en tu navegador.

¿Te gustaría que revisemos algún detalle adicional o que pasemos al siguiente checklist? 🚀

en el mismo power shell me aparece lo siguiente: Welcome to Streamlit! If you’d like to receive helpful onboarding emails, news, offers, promotions, and the occasional swag, please enter your email address below. Otherwise, leave this field blank. Email: pero esto no es un entorno visual aun cierto?, esto es el backend aun?

Correcto, Roberto. Esa pantalla en PowerShell es parte del inicio de Streamlit, donde te ofrece registrarte con tu correo electrónico para recibir actualizaciones y recursos relacionados con la plataforma. Esto no es aún el entorno visual de tu aplicación; es simplemente un mensaje de bienvenida generado por Streamlit en el backend.

Una vez que completes esta etapa o dejes el campo de correo electrónico en blanco y presiones Enter, Streamlit procederá a ejecutar tu aplicación y abrirá automáticamente el entorno visual en tu navegador.

**Qué hacer ahora:**

1. **En la PowerShell:**
   * Si no deseas ingresar tu correo electrónico, simplemente deja el campo vacío y presiona Enter. Esto no afectará tu aplicación.
2. **Streamlit abrirá el entorno visual:**
   * Después de este paso, deberías ver un mensaje en la terminal indicando que Streamlit está ejecutando tu aplicación. Aparecerá un enlace similar a:
   * Local URL: http://localhost:8501
   * Streamlit abrirá automáticamente tu navegador predeterminado para mostrar tu interfaz visual.
3. **Si no se abre automáticamente:**
   * Copia y pega el enlace (http://localhost:8501) en tu navegador para acceder al entorno visual.

Ahora deberías ver tu aplicación de checklist funcionando. ¿Te gustaría ayuda para navegar el entorno visual o ajustar algo en el formulario? 😊

si, me gustaria ver la posibilidad de separar donde dice Marca y modelo, y tenerlo aparte. ademas de un buscador para ser mas facil encontrar la informacion. ahora la consulta, me abrio altiro un checklist de ingreso, habra la opcion para que el solicitante elija que tipo de checklist va a ocupar y lo otro, al momento de indicar la cantidad de equipos a ingresar, al momento de poner la serie , marca y modelo, se agregara la informacion segun la solicitud?

¡Excelente idea, Roberto! Hagamos los ajustes para que tu aplicación sea más clara y funcional.

**Separar Marca y Modelo**

En lugar de combinar **marca** y **modelo** en una única columna desplegable, podemos separarlos en dos selectores independientes (uno para marca y otro para modelo). Además, podemos implementar un buscador para hacer la selección más dinámica.

**Opción para seleccionar el tipo de checklist**

Podemos añadir una pantalla inicial donde el solicitante elija el tipo de checklist que necesita (ingreso, egreso o movimiento interno). Según la elección, se cargará el formulario correspondiente.

**Cantidad de equipos y agregar información**

Cuando el solicitante indique la cantidad de equipos, podemos mostrar campos dinámicos para cada equipo. Esto incluirá series, marcas y modelos, asegurando que toda la información se asocie correctamente a la solicitud.

**Código actualizado para implementar estas funciones**

**Pantalla inicial para elegir tipo de checklist**

Agrega esto al principio de app.py:

python

import streamlit as st

import sqlite3

import pandas as pd

# Pantalla inicial para elegir tipo de checklist

st.title("Checklist de Data Center")

tipo\_checklist = st.selectbox("Seleccione el tipo de checklist:", ["Ingreso", "Egreso", "Movimiento interno"])

if tipo\_checklist == "Ingreso":

st.subheader("Checklist de ingreso")

elif tipo\_checklist == "Egreso":

st.subheader("Checklist de egreso")

elif tipo\_checklist == "Movimiento interno":

st.subheader("Checklist de movimiento interno")

Esto mostrará una opción inicial para seleccionar el tipo de checklist. Luego, dependiendo de la elección, se cargará el formulario correspondiente.

**Separación de marca y modelo con buscador**

Actualiza el formulario en el caso de ingreso para que los campos de marca y modelo estén separados y dinámicos.

python

# Obtener catálogo de marcas y modelos

def obtener\_marcas():

conn = sqlite3.connect("checklist\_data\_center.db")

marcas\_df = pd.read\_sql\_query("SELECT DISTINCT marca FROM catalogo", conn)

conn.close()

return marcas\_df["marca"].tolist()

def obtener\_modelos(marca\_seleccionada):

conn = sqlite3.connect("checklist\_data\_center.db")

modelos\_df = pd.read\_sql\_query("SELECT modelo FROM catalogo WHERE marca = ?", conn, params=(marca\_seleccionada,))

conn.close()

return modelos\_df["modelo"].tolist()

# Separación de Marca y Modelo

marca\_seleccionada = st.selectbox("Seleccione la marca del equipo:", obtener\_marcas())

modelo\_seleccionado = st.selectbox("Seleccione el modelo del equipo:", obtener\_modelos(marca\_seleccionada))

# Mostrar la selección al usuario

st.write(f"Marca seleccionada: \*\*{marca\_seleccionada}\*\*")

st.write(f"Modelo seleccionado: \*\*{modelo\_seleccionado}\*\*")

Este código:

* Crea dos selectores separados: uno para marca y otro para modelo.
* Filtra los modelos disponibles según la marca seleccionada, utilizando consultas SQL dinámicas.

**Campos dinámicos según la cantidad de equipos**

Actualiza el formulario para generar campos dinámicos según la cantidad de equipos que ingresará el solicitante.

python

cantidad\_equipos = st.number\_input("Cantidad de equipos a ingresar:", min\_value=1, step=1)

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

marca\_seleccionada = st.selectbox(f"Marca del equipo {i}:", obtener\_marcas(), key=f"marca\_{i}")

modelo\_seleccionado = st.selectbox(f"Modelo del equipo {i}:", obtener\_modelos(marca\_seleccionada), key=f"modelo\_{i}")

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

equipos.append({"marca": marca\_seleccionada, "modelo": modelo\_seleccionado, "serie": numero\_serie})

# Guardar información

if st.button("Guardar solicitud"):

conn = sqlite3.connect("checklist\_data\_center.db")

cursor = conn.cursor()

for equipo in equipos:

cursor.execute("""

INSERT INTO solicitudes\_ingreso (

solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

marca, modelo, numero\_serie

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

equipo["marca"], equipo["modelo"], equipo["serie"]))

conn.commit()

conn.close()

st.success("¡Solicitud de ingreso guardada correctamente!")

Este código:

* Genera múltiples campos dinámicos según la cantidad de equipos indicada.
* Guarda todos los datos en la tabla solicitudes\_ingreso, asociando la información de cada equipo con la solicitud.

**Prueba de la funcionalidad**

1. Ejecuta streamlit run app.py.
2. Verifica que el solicitante pueda:
   * Elegir el tipo de checklist al inicio.
   * Separar marca y modelo de forma dinámica con el buscador.
   * Llenar información para varios equipos según la cantidad indicada.

**Siguientes pasos**

* Validar que los datos se guarden correctamente en la tabla correspondiente.
* Ajustar las interfaces para los checklists de egreso y movimiento interno.
* Crear una sección para que el equipo del Data Center gestione las solicitudes finalizadas.

¿Quieres que avancemos con alguna de estas funcionalidades o probemos el código ajustado? 😊

los codigos que me indicastes donde deben ir, solo el primero me indicaste que iba enm app.py y los otros?

¡Buena observación, Roberto! Déjame detallarte dónde deben ir cada uno de los fragmentos de código para mantener tu proyecto bien organizado y funcional.

**1. Pantalla inicial para elegir el tipo de checklist**

Este código debe ir **al inicio** del archivo app.py. Es el punto de entrada de la aplicación, que permite al usuario elegir entre las opciones "Ingreso", "Egreso" o "Movimiento interno".

**2. Funciones para obtener marcas y modelos**

Estas funciones son utilitarias y se encargan de interactuar con la base de datos para obtener las marcas y modelos dinámicamente. Es mejor organizarlas en el mismo archivo app.py, **antes de la interfaz visual**, como parte de las funciones auxiliares.

Pon estas funciones justo debajo de la función conectar\_db() (si ya existe) o al inicio del archivo, después de los imports:

python

# Función para obtener marcas únicas desde el catálogo

def obtener\_marcas():

conn = sqlite3.connect("checklist\_data\_center.db")

marcas\_df = pd.read\_sql\_query("SELECT DISTINCT marca FROM catalogo", conn)

conn.close()

return marcas\_df["marca"].tolist()

# Función para obtener modelos en función de la marca seleccionada

def obtener\_modelos(marca\_seleccionada):

conn = sqlite3.connect("checklist\_data\_center.db")

modelos\_df = pd.read\_sql\_query("SELECT modelo FROM catalogo WHERE marca = ?", conn, params=(marca\_seleccionada,))

conn.close()

return modelos\_df["modelo"].tolist()

**3. Separar marca y modelo con selectores dinámicos**

Este código debe ir en la sección de la interfaz visual, **dentro del bloque que maneja el checklist de ingreso**. Por ejemplo:

python

if tipo\_checklist == "Ingreso":

st.subheader("Checklist de ingreso")

# Coloca el código para separar marca y modelo aquí

marca\_seleccionada = st.selectbox("Seleccione la marca del equipo:", obtener\_marcas())

modelo\_seleccionado = st.selectbox("Seleccione el modelo del equipo:", obtener\_modelos(marca\_seleccionada))

Esto asegura que los campos para elegir marca y modelo se muestren solo cuando el solicitante selecciona el **checklist de ingreso**.

**4. Campos dinámicos según cantidad de equipos**

Este bloque de código también pertenece a la sección del **checklist de ingreso**, justo **después** de los datos principales (como nombre del solicitante, fecha y cliente). Este código debe calcular dinámicamente los equipos en función de la cantidad ingresada:

python

cantidad\_equipos = st.number\_input("Cantidad de equipos a ingresar:", min\_value=1, step=1)

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

marca\_seleccionada = st.selectbox(f"Marca del equipo {i}:", obtener\_marcas(), key=f"marca\_{i}")

modelo\_seleccionado = st.selectbox(f"Modelo del equipo {i}:", obtener\_modelos(marca\_seleccionada), key=f"modelo\_{i}")

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

equipos.append({"marca": marca\_seleccionada, "modelo": modelo\_seleccionado, "serie": numero\_serie})

**5. Botón para guardar la solicitud**

El botón que guarda los datos debe ir al final del formulario del **checklist de ingreso**, justo después de la información dinámica de los equipos. Si prefieres mantenerlo separado, puede quedar en su propio bloque:

python

if st.button("Guardar solicitud"):

conn = sqlite3.connect("checklist\_data\_center.db")

cursor = conn.cursor()

for equipo in equipos:

cursor.execute("""

INSERT INTO solicitudes\_ingreso (

solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

marca, modelo, numero\_serie

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

equipo["marca"], equipo["modelo"], equipo["serie"]))

conn.commit()

conn.close()

st.success("¡Solicitud de ingreso guardada correctamente!")

**Resumen de la estructura del archivo app.py**

Tu archivo app.py quedará estructurado de esta forma:

app.py

├── Importación de librerías

├── Funciones auxiliares (e.g., conectar\_db, obtener\_marcas, obtener\_modelos)

├── Pantalla inicial para elegir el tipo de checklist

├── Lógica del checklist de ingreso

│ ├── Datos básicos del solicitante

│ ├── Selectores de marca y modelo

│ ├── Campos dinámicos según cantidad de equipos

│ └── Botón para guardar los datos

├── Lógica del checklist de egreso (por implementar)

└── Lógica del checklist de movimiento interno (por implementar)

**Siguientes pasos**

1. Implementa estos cambios y valida que las funciones se ejecuten en la parte correcta de la aplicación.
2. Si quieres, podemos pasar a trabajar en los checklists de **egreso** y **movimiento interno** o profundizar en la funcionalidad del equipo del Data Center para gestionar estas solicitudes.

Avísame cómo te va y seguimos mejorando juntos 🚀. ¡Este proyecto está tomando forma genial! 😊

me darias el codigo definitivo de la app.py ya me enrede

¡Entendido, Roberto! Vamos a organizar todo en un archivo **app.py** definitivo y completo para que no haya confusión. Este archivo incluirá la lógica para el checklist de ingreso que hemos desarrollado hasta ahora, con las mejoras como separar marca y modelo, elegir el tipo de checklist, y manejar varios equipos según la cantidad ingresada.

**Código completo para app.py**

python

import streamlit as st

import sqlite3

import pandas as pd

# Función para conectar a la base de datos

def conectar\_db():

conn = sqlite3.connect("checklist\_data\_center.db")

return conn

# Función para obtener marcas únicas desde el catálogo

def obtener\_marcas():

conn = conectar\_db()

marcas\_df = pd.read\_sql\_query("SELECT DISTINCT marca FROM catalogo", conn)

conn.close()

return marcas\_df["marca"].tolist()

# Función para obtener modelos en función de la marca seleccionada

def obtener\_modelos(marca\_seleccionada):

conn = conectar\_db()

modelos\_df = pd.read\_sql\_query("SELECT modelo FROM catalogo WHERE marca = ?", conn, params=(marca\_seleccionada,))

conn.close()

return modelos\_df["modelo"].tolist()

# Función para obtener la lista de clientes

def obtener\_clientes():

# Si tienes otra fuente (como Excel) para clientes, conecta aquí.

return ["Cliente A", "Cliente B", "Cliente C"]

# Pantalla inicial para elegir el tipo de checklist

st.title("Checklist de Data Center")

tipo\_checklist = st.selectbox("Seleccione el tipo de checklist:", ["Ingreso", "Egreso", "Movimiento interno"])

# Lógica para el checklist de ingreso

if tipo\_checklist == "Ingreso":

st.subheader("Checklist de ingreso")

# Datos básicos del solicitante

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_ingreso = st.date\_input("Fecha de ingreso:")

cliente = st.selectbox("Seleccione el cliente:", obtener\_clientes())

datacenter = st.selectbox("Seleccione el DataCenter:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

ticket = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):")

# Cantidad de equipos a ingresar

cantidad\_equipos = st.number\_input("Cantidad de equipos a ingresar:", min\_value=1, step=1)

# Campos dinámicos para cada equipo

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

marca\_seleccionada = st.selectbox(f"Marca del equipo {i}:", obtener\_marcas(), key=f"marca\_{i}")

modelo\_seleccionado = st.selectbox(f"Modelo del equipo {i}:", obtener\_modelos(marca\_seleccionada), key=f"modelo\_{i}")

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

equipos.append({"marca": marca\_seleccionada, "modelo": modelo\_seleccionado, "serie": numero\_serie})

# Botón para guardar la solicitud

if st.button("Guardar solicitud"):

if all([solicitante, cliente, datacenter, ticket]) and all(e["serie"] for e in equipos):

conn = conectar\_db()

cursor = conn.cursor()

for equipo in equipos:

cursor.execute("""

INSERT INTO solicitudes\_ingreso (

solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

marca, modelo, numero\_serie

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

equipo["marca"], equipo["modelo"], equipo["serie"]))

conn.commit()

conn.close()

st.success("¡Solicitud de ingreso guardada correctamente!")

else:

st.error("Por favor, complete todos los campos obligatorios.")

# Espacio para lógica de checklist de egreso o movimiento interno (por implementar)

elif tipo\_checklist == "Egreso":

st.subheader("Checklist de egreso")

st.write("Esta sección aún está en desarrollo.")

elif tipo\_checklist == "Movimiento interno":

st.subheader("Checklist de movimiento interno")

st.write("Esta sección aún está en desarrollo.")

**Explicación del código**

1. **Pantalla inicial para seleccionar el tipo de checklist:**
   * Un menú desplegable (st.selectbox) permite al usuario elegir entre los tipos de checklist: "Ingreso", "Egreso" y "Movimiento interno".
2. **Checklist de ingreso:**
   * Captura los datos básicos del solicitante, como nombre, fecha, cliente y DataCenter.
   * Permite especificar la cantidad de equipos y genera dinámicamente los campos para cada equipo (marca, modelo, número de serie).
   * Los datos de marca y modelo se cargan dinámicamente desde la base de datos (catalogo) utilizando las funciones obtener\_marcas() y obtener\_modelos().
3. **Guardar los datos en la base de datos:**
   * Al presionar el botón "Guardar solicitud", la información ingresada por el usuario se guarda en la tabla solicitudes\_ingreso.
   * Se valida que todos los campos obligatorios estén completos antes de guardar.
4. **Placeholders para "Egreso" y "Movimiento interno":**
   * Estas secciones aún no están implementadas, pero el menú inicial las tiene listas para futuras mejoras.

**Pasos para probar**

1. Guarda este código como app.py en la carpeta principal de tu proyecto.
2. Abre la terminal, activa el entorno virtual (venv) y ejecuta el comando:

bash

streamlit run app.py

1. Verifica que:
   * Puedas seleccionar el tipo de checklist (en este caso, "Ingreso").
   * Completar los datos básicos del solicitante.
   * Generar campos dinámicos según la cantidad de equipos.
   * Seleccionar marca y modelo de forma separada.
   * Guardar los datos correctamente en la base de datos.

**Siguientes pasos**

* Implementar las secciones de "Egreso" y "Movimiento interno".
* Crear una interfaz para que el equipo del Data Center gestione las solicitudes y complete los detalles finales.

¡Déjame saber si tienes dudas o quieres continuar con algún ajuste o mejora adicional! 🚀😊

como salgo del streamlit?

Para salir de Streamlit y detener la aplicación que se está ejecutando, sigue estos pasos:

1. Ve a la terminal o consola donde ejecutaste el comando streamlit run app.py.
2. En esa misma terminal, presiona las teclas **Ctrl + C** (mantén presionada la tecla "Ctrl" y luego presiona "C").
   * Esto detendrá el servidor local de Streamlit y cerrará la aplicación.
3. Verás un mensaje en la terminal indicando que la ejecución ha sido interrumpida.

Si deseas reiniciar Streamlit más tarde, simplemente vuelve a ejecutar el comando streamlit run app.py. ¡Fácil y directo! 😊

sqlite3.OperationalError: table solicitudes\_ingreso has no column named marca Traceback: File "C:\Users\Rober\OneDrive\Escritorio\Checklist\_DataCenter\venv\Lib\site-packages\streamlit\runtime\scriptrunner\exec\_code.py", line 121, in exec\_func\_with\_error\_handling result = func() File "C:\Users\Rober\OneDrive\Escritorio\Checklist\_DataCenter\venv\Lib\site-packages\streamlit\runtime\scriptrunner\script\_runner.py", line 640, in code\_to\_exec exec(code, module.\_\_dict\_\_) ~~~~^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Rober\OneDrive\Escritorio\Checklist\_DataCenter\app.py", line 62, in <module> cursor.execute(""" ~~~~~~~~~~~~~~^^^^ INSERT INTO solicitudes\_ingreso ( ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ ...<3 lines>... """, (solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos, ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ equipo["marca"], equipo["modelo"], equipo["serie"])) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ eso me sale al darle guardar solicitud

El error que aparece indica que la columna marca no existe en la tabla solicitudes\_ingreso de tu base de datos. Esto probablemente se debe a que cuando se creó la tabla en el archivo database.py, no se definió una columna llamada marca.

Vamos a solucionarlo paso a paso:

**1. Verificar la definición de la tabla solicitudes\_ingreso**

Abre tu archivo database.py y revisa la definición de la tabla solicitudes\_ingreso. Asegúrate de que las columnas marca, modelo y numero\_serie estén incluidas. La definición correcta debería verse más o menos así:

**Código para crear la tabla solicitudes\_ingreso:**

python

cursor.execute("""

CREATE TABLE IF NOT EXISTS solicitudes\_ingreso (

id INTEGER PRIMARY KEY AUTOINCREMENT,

solicitante TEXT NOT NULL,

fecha\_ingreso TEXT NOT NULL,

cliente TEXT NOT NULL,

datacenter TEXT NOT NULL,

ticket TEXT NOT NULL,

cantidad\_equipos INTEGER NOT NULL,

marca TEXT NOT NULL,

modelo TEXT NOT NULL,

numero\_serie TEXT NOT NULL

)

""")

Si estas columnas (marca, modelo, numero\_serie) no están en la definición, agrégalas.

**2. Modificar la base de datos**

Si la tabla ya fue creada y no contiene estas columnas, necesitarás modificarla. Lamentablemente, SQLite no permite modificar las tablas directamente para agregar columnas, así que tendrás que seguir estos pasos:

1. **Copia la información existente**:
   * Si la tabla ya contiene datos que deseas preservar, puedes exportarlos a un archivo temporal o una nueva tabla.
2. **Elimina la tabla actual**:
   * Borra la tabla solicitudes\_ingreso para reemplazarla con la definición correcta:

python

cursor.execute("DROP TABLE IF EXISTS solicitudes\_ingreso")

1. **Recrea la tabla con la nueva estructura**:
   * Después de eliminar la tabla, vuelve a crearla con las columnas necesarias (incluyendo marca, modelo, numero\_serie).
2. **(Opcional) Restaura los datos**:
   * Si exportaste datos previos, puedes volver a insertarlos manualmente en la nueva tabla.

**3. Volver a correr el script database.py**

Una vez que hayas actualizado la definición de la tabla solicitudes\_ingreso, guarda el archivo database.py y vuelve a ejecutarlo en la terminal para aplicar los cambios:

bash

python database.py

Esto recreará la tabla solicitudes\_ingreso con las columnas correctas.

**4. Probar el formulario en app.py**

Después de actualizar la estructura de la tabla, vuelve a ejecutar tu aplicación de Streamlit:

bash

streamlit run app.py

Intenta guardar una solicitud nuevamente para confirmar que la columna marca ahora está disponible y que el error desapareció.

**Consejo adicional**

Si deseas validar rápidamente las tablas y columnas en tu base de datos SQLite, puedes usar una herramienta como **DB Browser for SQLite**. Esta herramienta te permite explorar las tablas, columnas y datos de forma visual para confirmar que todo está configurado correctamente.

¿Te gustaría que te ayude a revisar o ajustar algo más? ¡Avísame cómo te va con los cambios! 😊🚀

cursor.execute(""" CREATE TABLE IF NOT EXISTS solicitudes\_ingreso ( id INTEGER PRIMARY KEY AUTOINCREMENT, solicitante TEXT NOT NULL, fecha\_ingreso TEXT NOT NULL, cliente TEXT NOT NULL, datacenter TEXT NOT NULL, ticket TEXT NOT NULL, cantidad\_equipos INTEGER NOT NULL, marca\_modelo TEXT NOT NULL, numero\_serie TEXT NOT NULL, tipo\_equipo TEXT NOT NULL, rack\_asignado TEXT, ubicacion\_rack\_u TEXT, cantidad\_u INTEGER, propiedad TEXT, sala TEXT, dual\_single TEXT, tag TEXT, mt2 TEXT, estado\_inicial TEXT, detalles TEXT ) """) esa es mi tabla de ingreso actual

Gracias por compartir la estructura de tu tabla, Roberto. Basado en el diseño actual de tu tabla solicitudes\_ingreso, el campo marca\_modelo se utiliza para almacenar tanto la marca como el modelo en un solo texto combinado. Sin embargo, el error ocurre porque el código en tu archivo app.py intenta guardar marca y modelo en columnas separadas, las cuales no existen en la tabla actual.

**Solución: Ajustar la tabla o el código**

**Opción 1: Modificar la estructura de la tabla**

Si deseas manejar **marca** y **modelo** como columnas separadas en lugar de combinarlas en marca\_modelo, actualiza tu tabla solicitudes\_ingreso para incluir estas columnas. Aquí está el código para recrear la tabla con los cambios:

python

cursor.execute("DROP TABLE IF EXISTS solicitudes\_ingreso")

cursor.execute("""

CREATE TABLE IF NOT EXISTS solicitudes\_ingreso (

id INTEGER PRIMARY KEY AUTOINCREMENT,

solicitante TEXT NOT NULL,

fecha\_ingreso TEXT NOT NULL,

cliente TEXT NOT NULL,

datacenter TEXT NOT NULL,

ticket TEXT NOT NULL,

cantidad\_equipos INTEGER NOT NULL,

marca TEXT NOT NULL,

modelo TEXT NOT NULL,

numero\_serie TEXT NOT NULL,

tipo\_equipo TEXT NOT NULL,

rack\_asignado TEXT,

ubicacion\_rack\_u TEXT,

cantidad\_u INTEGER,

propiedad TEXT,

sala TEXT,

dual\_single TEXT,

tag TEXT,

mt2 TEXT,

estado\_inicial TEXT,

detalles TEXT

)

""")

Después de actualizar la tabla, asegúrate de ejecutar nuevamente el archivo database.py para aplicar los cambios. Luego, ajusta tu código en app.py para que guarde marca y modelo en las nuevas columnas.

**Opción 2: Ajustar el código de la aplicación**

Si prefieres mantener marca\_modelo como un solo campo en la tabla, actualiza el código en app.py para guardar la información combinada (por ejemplo, "Marca - Modelo") en esa columna. Aquí está el ajuste:

python

# Ajustar cómo se guarda la información en 'marca\_modelo'

marca\_seleccionada = st.selectbox("Seleccione la marca del equipo:", obtener\_marcas(), key=f"marca\_{i}")

modelo\_seleccionado = st.selectbox("Seleccione el modelo del equipo:", obtener\_modelos(marca\_seleccionada), key=f"modelo\_{i}")

marca\_modelo = f"{marca\_seleccionada} - {modelo\_seleccionado"

# Guardar en la base de datos usando 'marca\_modelo'

cursor.execute("""

INSERT INTO solicitudes\_ingreso (

solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

marca\_modelo, numero\_serie, tipo\_equipo

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

marca\_modelo, equipo["serie"], tipo\_equipo))

**Recomendación**

La decisión entre las opciones depende de cómo quieras manejar la información:

* Si deseas separar **marca** y **modelo** en columnas distintas para facilitar la búsqueda y gestión futura, sigue la **Opción 1**.
* Si prefieres mantenerlas combinadas para simplificar la estructura de la tabla, sigue la **Opción 2**.

¿Te gustaría ayuda para implementar una de estas soluciones o probar los ajustes? ¡Vamos avanzando juntos! 😊🚀

al momento de buscar marca y modelo, podemos ordenar ambos datos de la A a la Z y adicionalmente agregar un buscador de marca y un buscador de modelo para mejor precision?

¡Claro que sí, Roberto! Ordenar las marcas y modelos alfabéticamente y agregar buscadores para cada uno es una excelente manera de mejorar la experiencia del usuario. Esto facilita la navegación y la búsqueda precisa de los datos en la interfaz.

Voy a detallar cómo puedes lograr esto:

**1. Ordenar marcas y modelos alfabéticamente**

Asegurémonos de que las consultas SQL que obtienen las marcas y modelos incluyan un ordenamiento de A a Z usando ORDER BY. Aquí está el código actualizado para las funciones auxiliares:

**Función para obtener marcas:**

python

def obtener\_marcas():

conn = sqlite3.connect("checklist\_data\_center.db")

marcas\_df = pd.read\_sql\_query("SELECT DISTINCT marca FROM catalogo ORDER BY marca ASC", conn)

conn.close()

return marcas\_df["marca"].tolist()

**Función para obtener modelos basados en la marca seleccionada:**

python

def obtener\_modelos(marca\_seleccionada):

conn = sqlite3.connect("checklist\_data\_center.db")

modelos\_df = pd.read\_sql\_query(

"SELECT modelo FROM catalogo WHERE marca = ? ORDER BY modelo ASC",

conn, params=(marca\_seleccionada,))

conn.close()

return modelos\_df["modelo"].tolist()

Estas consultas aseguran que las marcas y modelos estén siempre ordenados alfabéticamente.

**2. Agregar un buscador para marcas y modelos**

En la interfaz con **Streamlit**, puedes usar st.text\_input para crear buscadores dinámicos. Esto permite que los usuarios ingresen texto para filtrar las opciones de marcas y modelos.

**Código para el buscador de marcas y modelos:**

python

# Buscador de marcas

marca\_busqueda = st.text\_input("Buscar marca:")

marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()]

marca\_seleccionada = st.selectbox("Seleccione la marca del equipo:", marcas)

# Buscador de modelos

if marca\_seleccionada:

modelo\_busqueda = st.text\_input("Buscar modelo:")

modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()]

modelo\_seleccionado = st.selectbox("Seleccione el modelo del equipo:", modelos)

# Mostrar la selección al usuario

st.write(f"Marca seleccionada: \*\*{marca\_seleccionada}\*\*")

st.write(f"Modelo seleccionado: \*\*{modelo\_seleccionado}\*\*")

**3. Cómo funciona este código**

1. **Buscador de marcas**:
   * El usuario puede escribir parte del nombre de la marca en el campo de texto marca\_busqueda.
   * El código filtra la lista de marcas disponibles para mostrar solo aquellas que coincidan con el texto ingresado.
2. **Buscador de modelos**:
   * Una vez que el usuario selecciona una marca, puede buscar modelos específicos relacionados con esa marca.
   * El filtro dinámico se realiza en base al texto ingresado en modelo\_busqueda.
3. **Interactividad**:
   * A medida que el usuario escribe, las opciones disponibles para marcas y modelos se actualizan dinámicamente.

**4. Integración en el formulario**

Este código debe ir dentro del bloque que maneja los datos del equipo en el checklist de ingreso. Por ejemplo:

python

cantidad\_equipos = st.number\_input("Cantidad de equipos a ingresar:", min\_value=1, step=1)

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

# Buscador de marcas

marca\_busqueda = st.text\_input(f"Buscar marca (Equipo {i}):", key=f"marca\_busqueda\_{i}")

marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()]

marca\_seleccionada = st.selectbox(f"Seleccione la marca del equipo {i}:", marcas, key=f"marca\_{i}")

# Buscador de modelos

modelo\_busqueda = st.text\_input(f"Buscar modelo (Equipo {i}):", key=f"modelo\_busqueda\_{i}")

modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()]

modelo\_seleccionado = st.selectbox(f"Seleccione el modelo del equipo {i}:", modelos, key=f"modelo\_{i}")

# Número de serie

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

equipos.append({"marca": marca\_seleccionada, "modelo": modelo\_seleccionado, "serie": numero\_serie})

**5. Prueba del formulario**

1. Guarda el código y ejecuta:

bash

streamlit run app.py

1. Prueba las nuevas funcionalidades:
   * Busca y selecciona marcas y modelos con los nuevos buscadores.
   * Asegúrate de que los datos dinámicos se cargan correctamente según lo ingresado.

**Siguientes pasos**

* Verifica que los datos se guarden correctamente en la base de datos con la estructura actual.
* Si tienes alguna otra idea para mejorar la precisión o facilidad de uso, podemos implementarla.

¡Avísame cómo te va o si necesitas ajustar algo más! 🚀😊

me puedes dar el codigo definito con todos los cambios?

¡Por supuesto, Roberto! Aquí tienes el código definitivo de app.py, que incluye todos los cambios necesarios:

1. **Pantalla inicial para elegir tipo de checklist**: Permite al usuario seleccionar el tipo de formulario: ingreso, egreso o movimiento interno.
2. **Campos dinámicos según cantidad de equipos**: Genera múltiples campos para la información de equipos según la cantidad especificada.
3. **Separación de marca y modelo**: Con buscadores y selección ordenada alfabéticamente.
4. **Formulario de ingreso completo**: Incluye todos los campos obligatorios del solicitante y los equipos, con validación.

**Código definitivo para app.py**

python

import streamlit as st

import sqlite3

import pandas as pd

# Función para conectar a la base de datos

def conectar\_db():

conn = sqlite3.connect("checklist\_data\_center.db")

return conn

# Función para obtener marcas únicas desde el catálogo

def obtener\_marcas():

conn = conectar\_db()

marcas\_df = pd.read\_sql\_query("SELECT DISTINCT marca FROM catalogo ORDER BY marca ASC", conn)

conn.close()

return marcas\_df["marca"].tolist()

# Función para obtener modelos en función de la marca seleccionada

def obtener\_modelos(marca\_seleccionada):

conn = conectar\_db()

modelos\_df = pd.read\_sql\_query(

"SELECT modelo FROM catalogo WHERE marca = ? ORDER BY modelo ASC",

conn, params=(marca\_seleccionada,))

conn.close()

return modelos\_df["modelo"].tolist()

# Función para obtener clientes (puedes adaptarla si tienes otra fuente como Excel)

def obtener\_clientes():

return ["Cliente A", "Cliente B", "Cliente C"]

# Pantalla inicial para elegir el tipo de checklist

st.title("Checklist de Data Center")

tipo\_checklist = st.selectbox("Seleccione el tipo de checklist:", ["Ingreso", "Egreso", "Movimiento interno"])

# Lógica para el checklist de ingreso

if tipo\_checklist == "Ingreso":

st.subheader("Checklist de ingreso")

# Datos básicos del solicitante

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_ingreso = st.date\_input("Fecha de ingreso:")

cliente = st.selectbox("Seleccione el cliente:", obtener\_clientes())

datacenter = st.selectbox("Seleccione el DataCenter:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

ticket = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):")

# Cantidad de equipos a ingresar

cantidad\_equipos = st.number\_input("Cantidad de equipos a ingresar:", min\_value=1, step=1)

# Campos dinámicos para cada equipo

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

# Buscador de marcas

marca\_busqueda = st.text\_input(f"Buscar marca (Equipo {i}):", key=f"marca\_busqueda\_{i}")

marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()]

marca\_seleccionada = st.selectbox(f"Seleccione la marca del equipo {i}:", marcas, key=f"marca\_{i}")

# Buscador de modelos

modelo\_busqueda = st.text\_input(f"Buscar modelo (Equipo {i}):", key=f"modelo\_busqueda\_{i}")

modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()]

modelo\_seleccionado = st.selectbox(f"Seleccione el modelo del equipo {i}:", modelos, key=f"modelo\_{i}")

# Número de serie

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

equipos.append({"marca": marca\_seleccionada, "modelo": modelo\_seleccionado, "serie": numero\_serie})

# Botón para guardar la solicitud

if st.button("Guardar solicitud"):

if all([solicitante, cliente, datacenter, ticket]) and all(e["serie"] for e in equipos):

conn = conectar\_db()

cursor = conn.cursor()

for equipo in equipos:

cursor.execute("""

INSERT INTO solicitudes\_ingreso (

solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

marca\_modelo, numero\_serie, tipo\_equipo

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

f"{equipo['marca']} - {equipo['modelo']}", equipo["serie"], "Automático"))

conn.commit()

conn.close()

st.success("¡Solicitud de ingreso guardada correctamente!")

else:

st.error("Por favor, complete todos los campos obligatorios.")

# Espacio para lógica de checklist de egreso o movimiento interno (por implementar)

elif tipo\_checklist == "Egreso":

st.subheader("Checklist de egreso")

st.write("Esta sección aún está en desarrollo.")

elif tipo\_checklist == "Movimiento interno":

st.subheader("Checklist de movimiento interno")

st.write("Esta sección aún está en desarrollo.")

**Características incluidas**

1. **Buscadores de marca y modelo**:
   * Ordenados alfabéticamente.
   * Permiten al usuario escribir para filtrar los resultados.
2. **Campos dinámicos para equipos**:
   * Generan la cantidad exacta de campos según lo indicado por el usuario.
3. **Pantalla inicial**:
   * Selección del tipo de checklist con opciones predefinidas.
4. **Validación**:
   * Se asegura que todos los campos obligatorios estén completos antes de guardar la solicitud.
5. **Almacenamiento en la base de datos**:
   * Marca y modelo se guardan como un solo campo combinado en marca\_modelo.
   * Otros detalles, como el número de serie, se guardan correctamente.

**Prueba del código**

1. Guarda el archivo como app.py.
2. En la terminal, activa tu entorno virtual (venv) y ejecuta:

bash

streamlit run app.py

1. Prueba las funcionalidades:
   * Busca y selecciona marcas y modelos.
   * Indica la cantidad de equipos y completa los datos.
   * Guarda la información y verifica que los datos se guardan correctamente en la base de datos.

Si encuentras algún detalle que quieras ajustar o mejorar, avísame. ¡Este proyecto está quedando increíble! 🚀😊

File "C:\Users\Rober\OneDrive\Escritorio\Checklist\_DataCenter\venv\Lib\site-packages\streamlit\runtime\scriptrunner\exec\_code.py", line 121, in exec\_func\_with\_error\_handling result = func() File "C:\Users\Rober\OneDrive\Escritorio\Checklist\_DataCenter\venv\Lib\site-packages\streamlit\runtime\scriptrunner\script\_runner.py", line 640, in code\_to\_exec exec(code, module.\_\_dict\_\_) ~~~~^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Rober\OneDrive\Escritorio\Checklist\_DataCenter\app.py", line 74, in <module> cursor.execute(""" ~~~~~~~~~~~~~~^^^^ INSERT INTO solicitudes\_ingreso ( ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ ...<3 lines>... """, (solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos, ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ f"{equipo['marca']} - {equipo['modelo']}", equipo["serie"], "Automático")) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

El error indica que hay un problema en la ejecución del comando SQL cuando se intenta guardar los datos en la tabla solicitudes\_ingreso. Específicamente, parece que la columna tipo\_equipo está recibiendo el valor "Automático", pero la estructura de la tabla puede no estar configurada correctamente o puede haber discrepancias en los valores esperados.

**Posibles causas**

1. **La columna** tipo\_equipo **no está definida correctamente en la tabla.**
   * Asegúrate de que la tabla solicitudes\_ingreso incluye la columna tipo\_equipo y que está configurada para aceptar valores como "Automático".
2. **Desajuste en el comando SQL.**
   * El comando INSERT INTO solicitudes\_ingreso (...) puede estar intentando insertar más o menos valores que las columnas definidas en la tabla.
3. **Datos faltantes o valores incorrectos.**
   * Los datos que se están construyendo en el bucle for equipo in equipos pueden tener problemas, como campos vacíos o mal formados.

**Solución paso a paso**

**1. Verificar la estructura de la tabla solicitudes\_ingreso**

Abre tu archivo database.py y confirma que la tabla incluye las columnas definidas en el comando SQL. Debe verse algo así:

python

cursor.execute("""

CREATE TABLE IF NOT EXISTS solicitudes\_ingreso (

id INTEGER PRIMARY KEY AUTOINCREMENT,

solicitante TEXT NOT NULL,

fecha\_ingreso TEXT NOT NULL,

cliente TEXT NOT NULL,

datacenter TEXT NOT NULL,

ticket TEXT NOT NULL,

cantidad\_equipos INTEGER NOT NULL,

marca\_modelo TEXT NOT NULL,

numero\_serie TEXT NOT NULL,

tipo\_equipo TEXT NOT NULL,

rack\_asignado TEXT,

ubicacion\_rack\_u TEXT,

cantidad\_u INTEGER,

propiedad TEXT,

sala TEXT,

dual\_single TEXT,

tag TEXT,

mt2 TEXT,

estado\_inicial TEXT,

detalles TEXT

)

""")

Ejecuta database.py para asegurarte de que la estructura esté actualizada en la base de datos.

**2. Ajustar el comando SQL en app.py**

El comando SQL debe insertar valores en las columnas de manera consistente con la estructura de la tabla. Aquí tienes el ajuste:

python

cursor.execute("""

INSERT INTO solicitudes\_ingreso (

solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

marca\_modelo, numero\_serie, tipo\_equipo

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

f"{equipo['marca']} - {equipo['modelo']}", equipo["serie"], "Automático"))

Asegúrate de que las columnas especificadas en el comando coinciden exactamente con las definidas en la tabla.

**3. Verificar datos en el bucle for**

Asegúrate de que los datos en equipos sean válidos antes de insertar. Puedes agregar una verificación adicional:

python

for equipo in equipos:

if equipo["marca"] and equipo["modelo"] and equipo["serie"]: # Verificar que no hay valores vacíos

cursor.execute("""

INSERT INTO solicitudes\_ingreso (

solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

marca\_modelo, numero\_serie, tipo\_equipo

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

f"{equipo['marca']} - {equipo['modelo']}", equipo["serie"], "Automático"))

Esto previene que datos incompletos causen problemas en la inserción.

**4. Ejecutar y probar**

1. Corre nuevamente database.py para asegurarte de que la tabla esté actualizada.
2. Corre streamlit run app.py y prueba el formulario de ingreso.
3. Completa todos los datos obligatorios y verifica que los datos se guarden correctamente.

**Siguientes pasos**

Si el error persiste, revisemos juntos la estructura exacta de tu tabla y los datos generados en el formulario. ¡Avísame cómo te va! 😊🚀

cursor.execute(""" CREATE TABLE IF NOT EXISTS solicitudes\_ingreso ( id INTEGER PRIMARY KEY AUTOINCREMENT, solicitante TEXT NOT NULL, fecha\_ingreso TEXT NOT NULL, cliente TEXT NOT NULL, datacenter TEXT NOT NULL, ticket TEXT NOT NULL, cantidad\_equipos INTEGER NOT NULL, marca TEXT NOT NULL, modelo TEXT NOT NULL, numero\_serie TEXT NOT NULL, tipo\_equipo TEXT NOT NULL, rack\_asignado TEXT, ubicacion\_rack\_u TEXT, cantidad\_u INTEGER, propiedad TEXT, sala TEXT, dual\_single TEXT, tag TEXT, mt2 TEXT, estado\_inicial TEXT, detalles TEXT ) """) actualmente esta asi

Gracias por confirmar la estructura de la tabla solicitudes\_ingreso. Dado que ahora la tabla incluye las columnas **marca** y **modelo** separadas, necesitamos ajustar el comando INSERT INTO solicitudes\_ingreso en el archivo app.py para reflejar esta nueva estructura.

Aquí está el ajuste necesario:

**Código actualizado en app.py**

Reemplaza el comando de inserción en tu archivo app.py por el siguiente:

python

cursor.execute("""

INSERT INTO solicitudes\_ingreso (

solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

marca, modelo, numero\_serie, tipo\_equipo

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

equipo["marca"], equipo["modelo"], equipo["serie"], "Automático"))

**Explicación de los ajustes**

1. **Separación de marca y modelo:**
   * En lugar de combinar marca y modelo como en la columna marca\_modelo, ahora se guardan como columnas independientes.
   * El código toma los valores de equipo["marca"] y equipo["modelo"] directamente y los guarda por separado.
2. **Consistencia con la estructura de la tabla:**
   * El comando SQL ahora incluye las columnas **marca**, **modelo**, **numero\_serie**, y **tipo\_equipo**, que son las que define la tabla actual.
   * El valor "Automático" es asignado a la columna tipo\_equipo.

**Prueba del código**

1. Actualiza tu archivo app.py con este cambio.
2. Ejecuta nuevamente la aplicación usando:

bash

streamlit run app.py

1. Completa el formulario de ingreso:
   * Selecciona marcas y modelos para los equipos.
   * Guarda la solicitud.
2. Verifica que los datos se almacenen correctamente en la tabla solicitudes\_ingreso.

Si encuentras algún detalle adicional que necesitemos ajustar, no dudes en comentarlo. ¡Estamos afinando cada aspecto para que todo funcione a la perfección! 😊🚀

ha quedado perfecto. continuemos

¡Qué alegría saber que quedó perfecto, Roberto! 🎉 Sigamos avanzando con tu proyecto. Podemos enfocarnos en:

1. **Checklist de egreso:** Crear el formulario para egreso de equipos, con campos dinámicos y validación de datos.
2. **Checklist de movimiento interno:** Desarrollar el formulario para movimientos dentro del Data Center.
3. **Gestión por el equipo del Data Center:** Crear una interfaz para que ellos agreguen información como racks asignados, ubicación en el rack, estado del equipo, etc.
4. **Generación de reportes:** Implementar una función para generar reportes resumidos basados en las tablas de la base de datos.

¿En cuál de estas áreas te gustaría que nos enfoquemos primero? ¡Me entusiasma seguir trabajando contigo! 🚀😊

vamos punto por punto

Perfecto, Roberto. Empecemos con el **Checklist de Egreso**, como primer punto.

**Punto 1: Checklist de Egreso**

Este formulario permitirá registrar todos los detalles necesarios cuando los equipos sean retirados del Data Center. Vamos a construir una interfaz similar al checklist de ingreso, pero adaptada a los datos de egreso. Aquí están los pasos clave:

**Campos necesarios para el checklist de egreso:**

1. **Datos básicos del solicitante:**
   * Nombre del solicitante.
   * Fecha del egreso.
2. **Información del equipo:**
   * Marca y modelo (separados, con buscadores como hicimos en el checklist de ingreso).
   * Número de serie.
   * Rack de origen.
3. **Razón y estado del equipo:**
   * Motivo del egreso (desinstalación, traslado, etc.).
   * Estado del equipo al salir (funcional/dañado).
4. **Notas adicionales:**
   * Detalles adicionales o comentarios.

**Código para app.py - Checklist de Egreso**

Añade este bloque al archivo app.py en la sección donde manejamos el tipo de checklist seleccionado:

python

elif tipo\_checklist == "Egreso":

st.subheader("Checklist de egreso")

# Datos básicos del solicitante

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_egreso = st.date\_input("Fecha de egreso:")

# Información del equipo

cantidad\_equipos = st.number\_input("Cantidad de equipos a egresar:", min\_value=1, step=1)

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

# Buscador de marcas

marca\_busqueda = st.text\_input(f"Buscar marca (Equipo {i}):", key=f"marca\_busqueda\_{i}")

marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()]

marca\_seleccionada = st.selectbox(f"Seleccione la marca del equipo {i}:", marcas, key=f"marca\_{i}")

# Buscador de modelos

modelo\_busqueda = st.text\_input(f"Buscar modelo (Equipo {i}):", key=f"modelo\_busqueda\_{i}")

modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()]

modelo\_seleccionado = st.selectbox(f"Seleccione el modelo del equipo {i}:", modelos, key=f"modelo\_{i}")

# Número de serie

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

# Rack de origen

rack\_origen = st.text\_input(f"Rack de origen del equipo {i}:", key=f"rack\_{i}")

# Motivo y estado

motivo\_egreso = st.selectbox(f"Motivo del egreso (Equipo {i}):", ["Desinstalación", "Traslado", "Otro"], key=f"motivo\_{i}")

estado\_salida = st.selectbox(f"Estado del equipo al salir (Equipo {i}):", ["Funcional", "Dañado"], key=f"estado\_{i}")

equipos.append({

"marca": marca\_seleccionada,

"modelo": modelo\_seleccionado,

"serie": numero\_serie,

"rack\_origen": rack\_origen,

"motivo\_egreso": motivo\_egreso,

"estado\_salida": estado\_salida

})

# Botón para guardar la solicitud

if st.button("Guardar solicitud de egreso"):

if all([solicitante, fecha\_egreso]) and all(e["serie"] for e in equipos):

conn = conectar\_db()

cursor = conn.cursor()

for equipo in equipos:

cursor.execute("""

INSERT INTO solicitudes\_egreso (

solicitante, fecha\_egreso, marca, modelo, numero\_serie,

rack\_origen, motivo\_egreso, estado\_salida

) VALUES (?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_egreso, equipo["marca"], equipo["modelo"], equipo["serie"],

equipo["rack\_origen"], equipo["motivo\_egreso"], equipo["estado\_salida"]))

conn.commit()

conn.close()

st.success("¡Solicitud de egreso guardada correctamente!")

else:

st.error("Por favor, complete todos los campos obligatorios.")

**Prueba del Checklist de Egreso**

1. Guarda el código en el archivo app.py.
2. Ejecuta la aplicación:

bash

streamlit run app.py

1. Selecciona la opción **Egreso** en la pantalla inicial y prueba el formulario:
   * Ingresa los datos del solicitante y el equipo.
   * Verifica que los datos se guardan correctamente en la base de datos.

**Siguientes pasos**

Si todo está funcionando bien, podemos avanzar al siguiente punto: **Checklist de Movimiento Interno**. ¿Quieres probar esta funcionalidad primero o seguimos adelante? 😊🚀

en el checklist de egreso, puedes poner las mismas funcionalidades que tienen en el checklist de ingreso, como el orden, el buscador, etc.

¡Claro, Roberto! Vamos a incorporar las funcionalidades avanzadas del checklist de ingreso al checklist de egreso. Esto incluye:

1. **Orden alfabético:** Tanto para marcas como para modelos.
2. **Buscador dinámico:** Permite filtrar las marcas y modelos escribiendo texto para mayor precisión.

Aquí está el código actualizado para el **Checklist de Egreso**, asegurando que todas las funcionalidades del checklist de ingreso sean aplicadas:

**Código actualizado para el Checklist de Egreso**

python

elif tipo\_checklist == "Egreso":

st.subheader("Checklist de egreso")

# Datos básicos del solicitante

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_egreso = st.date\_input("Fecha de egreso:")

# Cantidad de equipos a egresar

cantidad\_equipos = st.number\_input("Cantidad de equipos a egresar:", min\_value=1, step=1)

# Campos dinámicos para cada equipo

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

# Buscador de marcas

marca\_busqueda = st.text\_input(f"Buscar marca (Equipo {i}):", key=f"marca\_busqueda\_{i}")

marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()]

marca\_seleccionada = st.selectbox(f"Seleccione la marca del equipo {i}:", marcas, key=f"marca\_{i}")

# Buscador de modelos

modelo\_busqueda = st.text\_input(f"Buscar modelo (Equipo {i}):", key=f"modelo\_busqueda\_{i}")

modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()]

modelo\_seleccionado = st.selectbox(f"Seleccione el modelo del equipo {i}:", modelos, key=f"modelo\_{i}")

# Número de serie

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

# Rack de origen

rack\_origen = st.text\_input(f"Rack de origen del equipo {i}:", key=f"rack\_{i}")

# Motivo del egreso

motivo\_egreso = st.selectbox(f"Motivo del egreso (Equipo {i}):", ["Desinstalación", "Traslado", "Otro"], key=f"motivo\_{i}")

# Estado del equipo

estado\_salida = st.selectbox(f"Estado del equipo al salir (Equipo {i}):", ["Funcional", "Dañado"], key=f"estado\_{i}")

# Añadir equipo a la lista

equipos.append({

"marca": marca\_seleccionada,

"modelo": modelo\_seleccionado,

"serie": numero\_serie,

"rack\_origen": rack\_origen,

"motivo\_egreso": motivo\_egreso,

"estado\_salida": estado\_salida

})

# Botón para guardar la solicitud

if st.button("Guardar solicitud de egreso"):

if all([solicitante, fecha\_egreso]) and all(e["serie"] for e in equipos):

conn = conectar\_db()

cursor = conn.cursor()

for equipo in equipos:

cursor.execute("""

INSERT INTO solicitudes\_egreso (

solicitante, fecha\_egreso, marca, modelo, numero\_serie,

rack\_origen, motivo\_egreso, estado\_salida

) VALUES (?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_egreso, equipo["marca"], equipo["modelo"], equipo["serie"],

equipo["rack\_origen"], equipo["motivo\_egreso"], equipo["estado\_salida"]))

conn.commit()

conn.close()

st.success("¡Solicitud de egreso guardada correctamente!")

else:

st.error("Por favor, complete todos los campos obligatorios.")

**Qué se agregó**

1. **Orden alfabético de marcas y modelos:**
   * Las funciones obtener\_marcas y obtener\_modelos ya manejan el orden alfabético con ORDER BY.
2. **Buscadores dinámicos:**
   * Los buscadores permiten filtrar las marcas y modelos escribiendo en campos de texto (st.text\_input).
3. **Validación:**
   * El sistema verifica que todos los campos obligatorios estén completos antes de guardar.

**Prueba del Checklist de Egreso**

1. Guarda el código actualizado en app.py.
2. Ejecuta la aplicación:

bash

streamlit run app.py

1. Selecciona la opción **Egreso** y prueba las funcionalidades:
   * Usa los buscadores para filtrar marcas y modelos.
   * Completa los datos para varios equipos.
   * Guarda la solicitud y verifica los datos en la base de datos.

**Siguiente paso**

¿Está todo funcionando correctamente? Si quieres, podemos pasar al **Checklist de Movimiento Interno** o hacer ajustes adicionales en esta sección. ¡Seguimos avanzando juntos, Roberto! 🚀😊

me darias el codigo completo con este nuevo codigo agregado?

¡Por supuesto, Roberto! Aquí tienes el código **completo** para tu archivo app.py, que incluye tanto el **Checklist de Ingreso** como el **Checklist de Egreso** con todas las funcionalidades avanzadas: orden alfabético, buscadores dinámicos y validación de datos.

**Código completo para app.py**

python

import streamlit as st

import sqlite3

import pandas as pd

# Función para conectar a la base de datos

def conectar\_db():

conn = sqlite3.connect("checklist\_data\_center.db")

return conn

# Función para obtener marcas únicas desde el catálogo

def obtener\_marcas():

conn = conectar\_db()

marcas\_df = pd.read\_sql\_query("SELECT DISTINCT marca FROM catalogo ORDER BY marca ASC", conn)

conn.close()

return marcas\_df["marca"].tolist()

# Función para obtener modelos en función de la marca seleccionada

def obtener\_modelos(marca\_seleccionada):

conn = conectar\_db()

modelos\_df = pd.read\_sql\_query(

"SELECT modelo FROM catalogo WHERE marca = ? ORDER BY modelo ASC",

conn, params=(marca\_seleccionada,))

conn.close()

return modelos\_df["modelo"].tolist()

# Función para obtener clientes (puedes adaptarla si tienes otra fuente como Excel)

def obtener\_clientes():

return ["Cliente A", "Cliente B", "Cliente C"]

# Pantalla inicial para elegir el tipo de checklist

st.title("Checklist de Data Center")

tipo\_checklist = st.selectbox("Seleccione el tipo de checklist:", ["Ingreso", "Egreso", "Movimiento interno"])

# Lógica para el checklist de ingreso

if tipo\_checklist == "Ingreso":

st.subheader("Checklist de ingreso")

# Datos básicos del solicitante

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_ingreso = st.date\_input("Fecha de ingreso:")

cliente = st.selectbox("Seleccione el cliente:", obtener\_clientes())

datacenter = st.selectbox("Seleccione el DataCenter:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

ticket = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):")

# Cantidad de equipos a ingresar

cantidad\_equipos = st.number\_input("Cantidad de equipos a ingresar:", min\_value=1, step=1)

# Campos dinámicos para cada equipo

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

# Buscador de marcas

marca\_busqueda = st.text\_input(f"Buscar marca (Equipo {i}):", key=f"marca\_busqueda\_{i}")

marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()]

marca\_seleccionada = st.selectbox(f"Seleccione la marca del equipo {i}:", marcas, key=f"marca\_{i}")

# Buscador de modelos

modelo\_busqueda = st.text\_input(f"Buscar modelo (Equipo {i}):", key=f"modelo\_busqueda\_{i}")

modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()]

modelo\_seleccionado = st.selectbox(f"Seleccione el modelo del equipo {i}:", modelos, key=f"modelo\_{i}")

# Número de serie

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

equipos.append({"marca": marca\_seleccionada, "modelo": modelo\_seleccionado, "serie": numero\_serie})

# Botón para guardar la solicitud

if st.button("Guardar solicitud de ingreso"):

if all([solicitante, cliente, datacenter, ticket]) and all(e["serie"] for e in equipos):

conn = conectar\_db()

cursor = conn.cursor()

for equipo in equipos:

cursor.execute("""

INSERT INTO solicitudes\_ingreso (

solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

marca, modelo, numero\_serie, tipo\_equipo

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

equipo["marca"], equipo["modelo"], equipo["serie"], "Automático"))

conn.commit()

conn.close()

st.success("¡Solicitud de ingreso guardada correctamente!")

else:

st.error("Por favor, complete todos los campos obligatorios.")

# Lógica para el checklist de egreso

elif tipo\_checklist == "Egreso":

st.subheader("Checklist de egreso")

# Datos básicos del solicitante

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_egreso = st.date\_input("Fecha de egreso:")

# Cantidad de equipos a egresar

cantidad\_equipos = st.number\_input("Cantidad de equipos a egresar:", min\_value=1, step=1)

# Campos dinámicos para cada equipo

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

# Buscador de marcas

marca\_busqueda = st.text\_input(f"Buscar marca (Equipo {i}):", key=f"marca\_busqueda\_{i}")

marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()]

marca\_seleccionada = st.selectbox(f"Seleccione la marca del equipo {i}:", marcas, key=f"marca\_{i}")

# Buscador de modelos

modelo\_busqueda = st.text\_input(f"Buscar modelo (Equipo {i}):", key=f"modelo\_busqueda\_{i}")

modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()]

modelo\_seleccionado = st.selectbox(f"Seleccione el modelo del equipo {i}:", modelos, key=f"modelo\_{i}")

# Número de serie

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

# Rack de origen

rack\_origen = st.text\_input(f"Rack de origen del equipo {i}:", key=f"rack\_{i}")

# Motivo del egreso

motivo\_egreso = st.selectbox(f"Motivo del egreso (Equipo {i}):", ["Desinstalación", "Traslado", "Otro"], key=f"motivo\_{i}")

# Estado del equipo

estado\_salida = st.selectbox(f"Estado del equipo al salir (Equipo {i}):", ["Funcional", "Dañado"], key=f"estado\_{i}")

equipos.append({

"marca": marca\_seleccionada,

"modelo": modelo\_seleccionado,

"serie": numero\_serie,

"rack\_origen": rack\_origen,

"motivo\_egreso": motivo\_egreso,

"estado\_salida": estado\_salida

})

# Botón para guardar la solicitud

if st.button("Guardar solicitud de egreso"):

if all([solicitante, fecha\_egreso]) and all(e["serie"] for e in equipos):

conn = conectar\_db()

cursor = conn.cursor()

for equipo in equipos:

cursor.execute("""

INSERT INTO solicitudes\_egreso (

solicitante, fecha\_egreso, marca, modelo, numero\_serie,

rack\_origen, motivo\_egreso, estado\_salida

) VALUES (?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_egreso, equipo["marca"], equipo["modelo"], equipo["serie"],

equipo["rack\_origen"], equipo["motivo\_egreso"], equipo["estado\_salida"]))

conn.commit()

conn.close()

st.success("¡Solicitud de egreso guardada correctamente!")

else:

st.error("Por favor, complete todos los campos obligatorios.")

# Espacio para lógica de movimiento interno (por implementar)

elif tipo\_checklist == "Movimiento interno":

st.subheader("Checklist de movimiento interno")

st.write("Esta sección aún está en desarrollo.")

**Características completas en este código**

1. **Checklist de Ingreso:**
   * Orden alfabético y buscadores para marcas y modelos.
   * Campos dinámicos según cantidad de equipos.
   * Validación y almacenamiento en la base de datos.
2. **Checklist de Egreso:**
   * Misma funcionalidad avanzada: orden, buscadores y campos dinámicos.
   * Permite registrar información detallada sobre los equipos egresados.
3. **Pantalla inicial:**
   * Opción para seleccionar el tipo de checklist (Ingreso, Egreso, Movimiento Interno).
4. **Futuras expansiones:**
   * Espacio preparado para agregar la lógica del movimiento interno.

**Prueba del código**

1. Guarda este archivo como app.py.
2. Ejecuta en la terminal:

bash

streamlit run app.py

1. Prueba las opciones de **Ingreso** y **Egreso**.
2. Verifica que los datos se guarden correctamente en sus respectivas tablas de la base de datos.

Si tienes más ideas o ajustes, ¡aquí estoy para ayudarte! 😊🚀

Traceback (most recent call last): File "C:\Users\Rober\OneDrive\Escritorio\Checklist\_DataCenter\venv\Lib\site-packages\streamlit\runtime\scriptrunner\exec\_code.py", line 121, in exec\_func\_with\_error\_handling result = func() File "C:\Users\Rober\OneDrive\Escritorio\Checklist\_DataCenter\venv\Lib\site-packages\streamlit\runtime\scriptrunner\script\_runner.py", line 640, in code\_to\_exec exec(code, module.\_\_dict\_\_) ~~~~^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Rober\OneDrive\Escritorio\Checklist\_DataCenter\app.py", line 140, in <module> cursor.execute(""" ~~~~~~~~~~~~~~^^^^ INSERT INTO solicitudes\_egreso ( ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ ...<3 lines>... """, (solicitante, fecha\_egreso, equipo["marca"], equipo["modelo"], equipo["serie"], ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ equipo["rack\_origen"], equipo["motivo\_egreso"], equipo["estado\_salida"])) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ sqlite3.OperationalError: table solicitudes\_egreso has no column named marca

El error indica que la tabla solicitudes\_egreso no tiene una columna llamada marca, aunque el código intenta insertar datos en ella. Esto puede deberse a que la tabla se definió sin incluir esa columna.

**Solución paso a paso**

**1. Verifica la estructura de la tabla solicitudes\_egreso**

Abre tu archivo database.py y revisa cómo se definió la tabla solicitudes\_egreso. Debe incluir las columnas mencionadas en el comando INSERT INTO, que en este caso son marca y modelo.

La tabla debería verse así:

python

cursor.execute("""

CREATE TABLE IF NOT EXISTS solicitudes\_egreso (

id INTEGER PRIMARY KEY AUTOINCREMENT,

solicitante TEXT NOT NULL,

fecha\_egreso TEXT NOT NULL,

marca TEXT NOT NULL,

modelo TEXT NOT NULL,

numero\_serie TEXT NOT NULL,

rack\_origen TEXT,

motivo\_egreso TEXT NOT NULL,

estado\_salida TEXT NOT NULL

)

""")

Si esta estructura no incluye las columnas marca y modelo, actualiza el código y elimina la tabla existente para recrearla con las columnas necesarias.

**2. Recrear la tabla solicitudes\_egreso**

SQLite no permite agregar columnas directamente a una tabla existente, así que necesitas eliminar la tabla actual y volver a crearla.

Actualiza tu archivo database.py con este código:

python

cursor.execute("DROP TABLE IF EXISTS solicitudes\_egreso")

cursor.execute("""

CREATE TABLE IF NOT EXISTS solicitudes\_egreso (

id INTEGER PRIMARY KEY AUTOINCREMENT,

solicitante TEXT NOT NULL,

fecha\_egreso TEXT NOT NULL,

marca TEXT NOT NULL,

modelo TEXT NOT NULL,

numero\_serie TEXT NOT NULL,

rack\_origen TEXT,

motivo\_egreso TEXT NOT NULL,

estado\_salida TEXT NOT NULL

)

""")

Ejecuta el archivo database.py para aplicar los cambios y recrear la tabla con las columnas correctas:

bash

python database.py

**3. Probar el formulario de egreso**

Una vez que la tabla esté actualizada, vuelve a ejecutar tu aplicación:

bash

streamlit run app.py

Selecciona el tipo de checklist **Egreso** y completa el formulario. Asegúrate de que la información se guarda correctamente en la base de datos.

**Siguientes pasos**

Si prefieres avanzar al siguiente punto (Checklist de Movimiento Interno) después de probar esta solución, podemos implementarlo. ¡Avísame cómo te va! 😊🚀

falto agregar el datacenter del egreso y ticket

¡Gracias por el recordatorio, Roberto! Vamos a actualizar el checklist de egreso para incluir los campos **DataCenter del egreso** y **Ticket asociado**, asegurándonos de que la funcionalidad sea consistente con el resto del formulario. Aquí tienes el código ajustado que incluye estos nuevos campos:

**Código actualizado para el Checklist de Egreso**

python

elif tipo\_checklist == "Egreso":

st.subheader("Checklist de egreso")

# Datos básicos del solicitante

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_egreso = st.date\_input("Fecha de egreso:")

datacenter\_egreso = st.selectbox("Seleccione el DataCenter de egreso:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

ticket\_egreso = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):")

# Cantidad de equipos a egresar

cantidad\_equipos = st.number\_input("Cantidad de equipos a egresar:", min\_value=1, step=1)

# Campos dinámicos para cada equipo

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

# Buscador de marcas

marca\_busqueda = st.text\_input(f"Buscar marca (Equipo {i}):", key=f"marca\_busqueda\_{i}")

marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()]

marca\_seleccionada = st.selectbox(f"Seleccione la marca del equipo {i}:", marcas, key=f"marca\_{i}")

# Buscador de modelos

modelo\_busqueda = st.text\_input(f"Buscar modelo (Equipo {i}):", key=f"modelo\_busqueda\_{i}")

modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()]

modelo\_seleccionado = st.selectbox(f"Seleccione el modelo del equipo {i}:", modelos, key=f"modelo\_{i}")

# Número de serie

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

# Rack de origen

rack\_origen = st.text\_input(f"Rack de origen del equipo {i}:", key=f"rack\_{i}")

# Motivo del egreso

motivo\_egreso = st.selectbox(f"Motivo del egreso (Equipo {i}):", ["Desinstalación", "Traslado", "Otro"], key=f"motivo\_{i}")

# Estado del equipo

estado\_salida = st.selectbox(f"Estado del equipo al salir (Equipo {i}):", ["Funcional", "Dañado"], key=f"estado\_{i}")

# Añadir equipo a la lista

equipos.append({

"marca": marca\_seleccionada,

"modelo": modelo\_seleccionado,

"serie": numero\_serie,

"rack\_origen": rack\_origen,

"motivo\_egreso": motivo\_egreso,

"estado\_salida": estado\_salida

})

# Botón para guardar la solicitud

if st.button("Guardar solicitud de egreso"):

if all([solicitante, fecha\_egreso, datacenter\_egreso, ticket\_egreso]) and all(e["serie"] for e in equipos):

conn = conectar\_db()

cursor = conn.cursor()

for equipo in equipos:

cursor.execute("""

INSERT INTO solicitudes\_egreso (

solicitante, fecha\_egreso, datacenter, ticket, marca, modelo, numero\_serie,

rack\_origen, motivo\_egreso, estado\_salida

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_egreso, datacenter\_egreso, ticket\_egreso, equipo["marca"], equipo["modelo"], equipo["serie"],

equipo["rack\_origen"], equipo["motivo\_egreso"], equipo["estado\_salida"]))

conn.commit()

conn.close()

st.success("¡Solicitud de egreso guardada correctamente!")

else:

st.error("Por favor, complete todos los campos obligatorios.")

**Qué se agregó**

1. **DataCenter del Egreso:**
   * Añadido un campo desplegable (st.selectbox) para que el solicitante pueda seleccionar el DataCenter relacionado con el egreso.
2. **Ticket asociado:**
   * Añadido un campo de texto (st.text\_input) para ingresar el número del ticket o marcarlo como "Pendiente" si aún no existe.
3. **Validación de campos:**
   * Se incluye datacenter\_egreso y ticket\_egreso como campos obligatorios en la validación antes de guardar los datos.
4. **Almacenamiento:**
   * El comando INSERT INTO se actualizó para incluir las columnas datacenter y ticket.

**Pasos para probar**

1. Guarda este código actualizado en app.py.
2. Ejecuta la aplicación:

bash

streamlit run app.py

1. Selecciona la opción **Egreso** en la pantalla inicial.
2. Completa el formulario:
   * Ingresa datos del solicitante, DataCenter, ticket, y equipos.
   * Guarda la solicitud.
3. Verifica que todos los datos (incluidos datacenter y ticket) se guardan correctamente en la base de datos.

¿Todo claro, Roberto? Si hay algo más que ajustar o quieres avanzar al siguiente punto, ¡házmelo saber! 😊🚀

hay que agregar los cambios a la base de datos, asi esta ahora cursor.execute(""" CREATE TABLE IF NOT EXISTS solicitudes\_egreso ( id INTEGER PRIMARY KEY AUTOINCREMENT, solicitante TEXT NOT NULL, fecha\_egreso TEXT NOT NULL, marca TEXT NOT NULL, modelo TEXT NOT NULL, numero\_serie TEXT NOT NULL, rack\_origen TEXT, motivo\_egreso TEXT NOT NULL, estado\_salida TEXT NOT NULL ) """)

Gracias por compartir la estructura actual de la tabla solicitudes\_egreso, Roberto. Vamos a añadir los campos que faltan: **datacenter** y **ticket**, tal como se mencionaron en el checklist de egreso. Esto permitirá que los nuevos datos se guarden correctamente.

**Código actualizado para database.py**

Debemos eliminar la tabla existente (si ya está creada) y volver a crearla con las nuevas columnas. Aquí está el código actualizado para redefinir la tabla solicitudes\_egreso:

python

cursor.execute("DROP TABLE IF EXISTS solicitudes\_egreso")

cursor.execute("""

CREATE TABLE IF NOT EXISTS solicitudes\_egreso (

id INTEGER PRIMARY KEY AUTOINCREMENT,

solicitante TEXT NOT NULL,

fecha\_egreso TEXT NOT NULL,

datacenter TEXT NOT NULL,

ticket TEXT NOT NULL,

marca TEXT NOT NULL,

modelo TEXT NOT NULL,

numero\_serie TEXT NOT NULL,

rack\_origen TEXT,

motivo\_egreso TEXT NOT NULL,

estado\_salida TEXT NOT NULL

)

""")

**Pasos para aplicar los cambios**

1. **Abre tu archivo** database.py:
   * Reemplaza la definición actual de la tabla solicitudes\_egreso con el código actualizado anterior.
   * Esto asegura que las nuevas columnas **datacenter** y **ticket** estén incluidas.
2. **Ejecuta** database.py:
   * En tu terminal, ejecuta:

bash

python database.py

* + Esto eliminará la tabla existente (si existe) y la recreará con la estructura actualizada.

1. **Prueba en la aplicación**:
   * Vuelve a ejecutar tu aplicación app.py:

bash

streamlit run app.py

* + Selecciona el checklist de egreso y completa el formulario con los datos de **datacenter** y **ticket**, junto con los otros campos.

**Resultado esperado**

La tabla solicitudes\_egreso ahora tendrá las siguientes columnas:

* **id**: Identificador único de cada registro.
* **solicitante**: Nombre de quien solicita el egreso.
* **fecha\_egreso**: Fecha de cuando ocurre el egreso.
* **datacenter**: DataCenter relacionado con el egreso.
* **ticket**: Número del ticket asociado (o "Pendiente").
* **marca**: Marca del equipo.
* **modelo**: Modelo del equipo.
* **numero\_serie**: Número de serie del equipo.
* **rack\_origen**: Rack desde el cual se retira el equipo.
* **motivo\_egreso**: Razón del egreso (desinstalación, traslado, etc.).
* **estado\_salida**: Estado del equipo al momento del egreso (funcional/dañado).

**Siguientes pasos**

Prueba la funcionalidad y confirma si los datos se guardan correctamente. Si todo está funcionando como esperas, podemos avanzar al siguiente punto: **Checklist de Movimiento Interno** o cualquier otra funcionalidad que desees priorizar. 🚀😊

tenemos 2 problemas. el primero que al momento de buscar un modelo, este al filtrar sale repetida 4 veces el mismo dato. y lo segundo, al momento de egresar un equipo, al ir a la base de datos desde data center aparecen 4 datos como datacenter, ticket, marca y modelo en la misma celda de DataCenter

Entiendo perfectamente, Roberto. Vamos a resolver estos dos problemas de manera organizada y detallada:

**Problema 1: Modelos repetidos al buscar**

Esto sucede porque probablemente hay entradas duplicadas en la tabla catalogo de la base de datos. Aunque los datos parecen idénticos, pueden haber sido registrados varias veces.

**Solución: Eliminar duplicados en la base de datos y mejorar la consulta**

1. **Revisar la tabla** catalogo **en la base de datos:**
   * Abre la base de datos en una herramienta como **DB Browser for SQLite**.
   * Ejecuta esta consulta para identificar duplicados:

sql

SELECT marca, modelo, COUNT(\*)

FROM catalogo

GROUP BY marca, modelo

HAVING COUNT(\*) > 1;

* + Esto mostrará las marcas y modelos que están duplicados.

1. **Eliminar los duplicados de la tabla:**
   * Para eliminar duplicados, puedes usar una consulta como esta:

sql

DELETE FROM catalogo

WHERE rowid NOT IN (

SELECT MIN(rowid)

FROM catalogo

GROUP BY marca, modelo

);

* + Esta consulta conserva solo una entrada única para cada combinación de marca y modelo.

1. **Ajustar la consulta en** app.py **para evitar duplicados en futuras búsquedas:**
   * Asegúrate de usar DISTINCT en las consultas para las funciones obtener\_modelos y obtener\_marcas:

python

def obtener\_modelos(marca\_seleccionada):

conn = conectar\_db()

modelos\_df = pd.read\_sql\_query(

"SELECT DISTINCT modelo FROM catalogo WHERE marca = ? ORDER BY modelo ASC",

conn, params=(marca\_seleccionada,))

conn.close()

return modelos\_df["modelo"].tolist()

**Problema 2: Valores combinados en una celda de datacenter**

Esto sucede porque en el comando SQL para insertar datos, las columnas están recibiendo múltiples valores concatenados en lugar de valores independientes. Esto puede deberse a un error en el comando INSERT INTO o en los datos que se están pasando.

**Solución: Revisar y corregir la inserción en app.py**

Revisemos el bloque que guarda los datos en solicitudes\_egreso. El comando debería ser algo como esto:

python

cursor.execute("""

INSERT INTO solicitudes\_egreso (

solicitante, fecha\_egreso, datacenter, ticket, marca, modelo, numero\_serie,

rack\_origen, motivo\_egreso, estado\_salida

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_egreso, datacenter\_egreso, ticket\_egreso,

equipo["marca"], equipo["modelo"], equipo["serie"],

equipo["rack\_origen"], equipo["motivo\_egreso"], equipo["estado\_salida"]))

**Pasos para corregir:**

1. **Verifica que los valores que se están pasando son correctos:**
   * Asegúrate de que datacenter\_egreso y ticket\_egreso estén llegando como valores independientes, no concatenados.
   * Puedes imprimir los valores justo antes del INSERT para depurarlos:

python

print(datacenter\_egreso, ticket\_egreso, equipo["marca"], equipo["modelo"])

1. **Revisar la tabla** solicitudes\_egreso **en la base de datos:**
   * Asegúrate de que las columnas estén definidas correctamente y correspondan a lo que intenta insertar el código.

**Pasos para probar ambas soluciones**

1. **Limpieza de duplicados:**
   * Ejecuta las consultas para eliminar duplicados en la tabla catalogo.
   * Reinicia la aplicación y verifica si el problema con los modelos repetidos se resolvió.
2. **Corregir el comando** INSERT INTO**:**
   * Implementa el ajuste en app.py y verifica que los datos se inserten en las columnas correctas.
3. **Prueba la funcionalidad completa:**
   * En el checklist de egreso, completa los datos de un equipo y guarda la solicitud.
   * Verifica en la base de datos que los valores estén almacenados correctamente.

¿Me confirmas si estas soluciones resuelven los problemas? ¡Si surge algo más, seguimos ajustando! 😊🚀

donde descargo db broses for sqlite

Puedes descargar **DB Browser for SQLite** desde su sitio oficial aquí. Este sitio ofrece versiones para Windows, macOS y Linux, además de opciones como instaladores estándar y versiones portátiles. También puedes encontrarlo en **SourceForge** aquí.

Si necesitas ayuda para instalarlo o configurarlo, ¡avísame! 😊🚀

como abro la base de datos con DB browser for sqlite

Abrir tu base de datos con **DB Browser for SQLite** es muy sencillo. Aquí tienes los pasos detallados:

**1. Abre DB Browser for SQLite**

* Una vez instalado, abre el programa.
* En Windows, lo encontrarás buscándolo en el menú de inicio como "DB Browser for SQLite".
* En macOS o Linux, estará en tu lista de aplicaciones.

**2. Carga tu archivo de base de datos**

1. **Haz clic en "Open Database"**:
   * Esto abrirá una ventana para explorar tus archivos.
2. **Navega hasta tu base de datos**:
   * Busca el archivo de tu base de datos. En tu caso, debe ser el archivo llamado checklist\_data\_center.db, que probablemente esté en tu carpeta de proyecto.
3. **Selecciona el archivo y haz clic en "Abrir"**:
   * Esto cargará tu base de datos en el programa.

**3. Explora tu base de datos**

* Una vez que la base de datos esté abierta, verás varias pestañas:
  + **"Browse Data" (Examinar datos):** Aquí puedes ver los registros actuales en tus tablas.
  + **"Edit Pragmas":** Ajustes avanzados de la base de datos (no recomendado para cambios sin conocimientos técnicos).
  + **"Execute SQL":** Desde aquí puedes escribir y ejecutar comandos SQL.
  + **"Structure":** Muestra la estructura de cada tabla en tu base de datos, como las columnas y sus tipos.

**4. Explora tus tablas**

1. Haz clic en la pestaña **"Browse Data"**.
2. En el menú desplegable (parte superior izquierda), selecciona la tabla que deseas explorar, como catalogo o solicitudes\_egreso.
3. Aquí podrás ver todos los datos de la tabla seleccionada.

**5. Edita o elimina registros si es necesario**

* Si necesitas limpiar duplicados o verificar datos:
  1. Ve a la pestaña **"Execute SQL"**.
  2. Escribe los comandos SQL que necesitas (como eliminar duplicados o hacer consultas).
  3. Presiona el botón **"Execute"** para ejecutar el comando.

Si tienes alguna duda o necesitas ayuda para ejecutar algo en particular en DB Browser, dime y te guío con más detalles. 🚀😊

el codigo final seria?

¡Por supuesto, Roberto! Aquí está el código final actualizado y completo para tu archivo app.py, incluyendo las correcciones para las funcionalidades del checklist de ingreso y egreso:

**Código definitivo para app.py**

python

import streamlit as st

import sqlite3

import pandas as pd

# Función para conectar a la base de datos

def conectar\_db():

conn = sqlite3.connect("checklist\_data\_center.db")

return conn

# Función para obtener marcas únicas desde el catálogo

def obtener\_marcas():

conn = conectar\_db()

marcas\_df = pd.read\_sql\_query("SELECT DISTINCT marca FROM catalogo ORDER BY marca ASC", conn)

conn.close()

return marcas\_df["marca"].tolist()

# Función para obtener modelos en función de la marca seleccionada

def obtener\_modelos(marca\_seleccionada):

conn = conectar\_db()

modelos\_df = pd.read\_sql\_query(

"SELECT DISTINCT modelo FROM catalogo WHERE marca = ? ORDER BY modelo ASC",

conn, params=(marca\_seleccionada,))

conn.close()

return modelos\_df["modelo"].tolist()

# Función para obtener clientes

def obtener\_clientes():

return ["Cliente A", "Cliente B", "Cliente C"]

# Pantalla inicial para elegir el tipo de checklist

st.title("Checklist de Data Center")

tipo\_checklist = st.selectbox("Seleccione el tipo de checklist:", ["Ingreso", "Egreso", "Movimiento interno"])

# Lógica para el checklist de ingreso

if tipo\_checklist == "Ingreso":

st.subheader("Checklist de ingreso")

# Datos básicos del solicitante

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_ingreso = st.date\_input("Fecha de ingreso:")

cliente = st.selectbox("Seleccione el cliente:", obtener\_clientes())

datacenter = st.selectbox("Seleccione el DataCenter:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

ticket = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):")

# Cantidad de equipos a ingresar

cantidad\_equipos = st.number\_input("Cantidad de equipos a ingresar:", min\_value=1, step=1)

# Campos dinámicos para cada equipo

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

# Buscador de marcas

marca\_busqueda = st.text\_input(f"Buscar marca (Equipo {i}):", key=f"marca\_busqueda\_{i}")

marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()]

marca\_seleccionada = st.selectbox(f"Seleccione la marca del equipo {i}:", marcas, key=f"marca\_{i}")

# Buscador de modelos

modelo\_busqueda = st.text\_input(f"Buscar modelo (Equipo {i}):", key=f"modelo\_busqueda\_{i}")

modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()]

modelo\_seleccionado = st.selectbox(f"Seleccione el modelo del equipo {i}:", modelos, key=f"modelo\_{i}")

# Número de serie

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

equipos.append({"marca": marca\_seleccionada, "modelo": modelo\_seleccionado, "serie": numero\_serie})

# Botón para guardar la solicitud

if st.button("Guardar solicitud de ingreso"):

if all([solicitante, cliente, datacenter, ticket]) and all(e["serie"] for e in equipos):

conn = conectar\_db()

cursor = conn.cursor()

for equipo in equipos:

cursor.execute("""

INSERT INTO solicitudes\_ingreso (

solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

marca, modelo, numero\_serie, tipo\_equipo

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

equipo["marca"], equipo["modelo"], equipo["serie"], "Automático"))

conn.commit()

conn.close()

st.success("¡Solicitud de ingreso guardada correctamente!")

else:

st.error("Por favor, complete todos los campos obligatorios.")

# Lógica para el checklist de egreso

elif tipo\_checklist == "Egreso":

st.subheader("Checklist de egreso")

# Datos básicos del solicitante

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_egreso = st.date\_input("Fecha de egreso:")

datacenter\_egreso = st.selectbox("Seleccione el DataCenter de egreso:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

ticket\_egreso = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):")

# Cantidad de equipos a egresar

cantidad\_equipos = st.number\_input("Cantidad de equipos a egresar:", min\_value=1, step=1)

# Campos dinámicos para cada equipo

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

# Buscador de marcas

marca\_busqueda = st.text\_input(f"Buscar marca (Equipo {i}):", key=f"marca\_busqueda\_{i}")

marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()]

marca\_seleccionada = st.selectbox(f"Seleccione la marca del equipo {i}:", marcas, key=f"marca\_{i}")

# Buscador de modelos

modelo\_busqueda = st.text\_input(f"Buscar modelo (Equipo {i}):", key=f"modelo\_busqueda\_{i}")

modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()]

modelo\_seleccionado = st.selectbox(f"Seleccione el modelo del equipo {i}:", modelos, key=f"modelo\_{i}")

# Número de serie

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

# Rack de origen

rack\_origen = st.text\_input(f"Rack de origen del equipo {i}:", key=f"rack\_{i}")

# Motivo del egreso

motivo\_egreso = st.selectbox(f"Motivo del egreso (Equipo {i}):", ["Desinstalación", "Traslado", "Otro"], key=f"motivo\_{i}")

# Estado del equipo

estado\_salida = st.selectbox(f"Estado del equipo al salir (Equipo {i}):", ["Funcional", "Dañado"], key=f"estado\_{i}")

equipos.append({

"marca": marca\_seleccionada,

"modelo": modelo\_seleccionado,

"serie": numero\_serie,

"rack\_origen": rack\_origen,

"motivo\_egreso": motivo\_egreso,

"estado\_salida": estado\_salida

})

# Botón para guardar la solicitud

if st.button("Guardar solicitud de egreso"):

if all([solicitante, fecha\_egreso, datacenter\_egreso, ticket\_egreso]) and all(e["serie"] for e in equipos):

conn = conectar\_db()

cursor = conn.cursor()

for equipo in equipos:

cursor.execute("""

INSERT INTO solicitudes\_egreso (

solicitante, fecha\_egreso, datacenter, ticket, marca, modelo, numero\_serie,

rack\_origen, motivo\_egreso, estado\_salida

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_egreso, datacenter\_egreso, ticket\_egreso,

equipo["marca"], equipo["modelo"], equipo["serie"],

equipo["rack\_origen"], equipo["motivo\_egreso"], equipo["estado\_salida"]))

conn.commit()

conn.close()

st.success("¡Solicitud de egreso guardada correctamente!")

else:

st.error("Por favor, complete todos los campos obligatorios.")

# Espacio para lógica de movimiento interno (por implementar)

elif tipo\_checklist == "Movimiento interno":

st.subheader("Checklist de movimiento interno")

st.write("Esta sección aún está en desarrollo.")

**Características del código final**

1. **Checklist de Ingreso:**
   * Orden alfabético y buscadores para marcas y modelos.
   * Campos dinámicos basados en la cantidad de equipos.
   * Validación y almacenamiento en la base de datos con todos los campos requeridos.
2. **Checklist de Egreso:**
   * Misma funcionalidad avanzada que el checklist de ingreso.
   * Campos adicionales: datacenter y ticket, con inserción en la base de datos.
3. **Pantalla inicial:**
   * Opción para seleccionar el tipo de checklist (Ingreso, Egreso, Movimiento Interno).
4. **Futuras expansiones:**
   * Espacio preparado para agregar la lógica del movimiento interno.

**Pasos finales**

1. Guarda este archivo como app.py.
2. Ejecuta en la terminal:

bash

streamlit run app.py

1. Prueba las opciones de **Ingreso** y

File "C:\Users\Rober\OneDrive\Escritorio\Checklist\_DataCenter\venv\Lib\site-packages\pandas\io\sql.py", line 2674, in execute cur.execute(sql, \*args) ~~~~~~~~~~~^^^^^^^^^^^^ sqlite3.OperationalError: database is locked The above exception was the direct cause of the following exception: Traceback (most recent call last): File "c:\Users\Rober\OneDrive\Escritorio\Checklist\_DataCenter\app.py", line 55, in <module> marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()] ~~~~~~~~~~~~~~^^ File "c:\Users\Rober\OneDrive\Escritorio\Checklist\_DataCenter\app.py", line 13, in obtener\_marcas marcas\_df = pd.read\_sql\_query("SELECT DISTINCT marca FROM catalogo ORDER BY marca ASC", conn) File "C:\Users\Rober\OneDrive\Escritorio\Checklist\_DataCenter\venv\Lib\site-packages\pandas\io\sql.py", line 526, in read\_sql\_query return pandas\_sql.read\_query( ~~~~~~~~~~~~~~~~~~~~~^ sql, ^^^^ ...<6 lines>... dtype\_backend=dtype\_backend, ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ ) ^ File "C:\Users\Rober\OneDrive\Escritorio\Checklist\_DataCenter\venv\Lib\site-packages\pandas\io\sql.py", line 2738, in read\_query cursor = self.execute(sql, params) File "C:\Users\Rober\OneDrive\Escritorio\Checklist\_DataCenter\venv\Lib\site-packages\pandas\io\sql.py", line 2686, in execute raise ex from exc pandas.errors.DatabaseError: Execution failed on sql 'SELECT DISTINCT marca FROM catalogo ORDER BY marca ASC': database is locked

El error sqlite3.OperationalError: database is locked ocurre cuando SQLite no puede acceder a la base de datos porque ya está siendo utilizada por otro proceso o conexión que no se ha cerrado correctamente.

Aquí tienes una guía para solucionar este problema:

**Causas comunes del error**

1. **Conexión no cerrada**:
   * Una conexión previa a la base de datos sigue abierta, bloqueando cualquier nueva operación.
2. **Acceso simultáneo**:
   * Otro proceso o parte del código está intentando acceder a la base de datos al mismo tiempo.
3. **Bloqueo debido a una transacción pendiente**:
   * Si no se hace un commit() o close() después de ejecutar una consulta, la base de datos queda bloqueada.

**Soluciones**

**1. Cierra las conexiones después de usarlas**

Revisa tus funciones obtener\_marcas y obtener\_modelos, y asegúrate de que todas las conexiones a la base de datos se cierran correctamente después de ejecutarse. Aquí está el código revisado:

python

def obtener\_marcas():

conn = conectar\_db()

try:

marcas\_df = pd.read\_sql\_query("SELECT DISTINCT marca FROM catalogo ORDER BY marca ASC", conn)

return marcas\_df["marca"].tolist()

finally:

conn.close() # Asegúrate de cerrar la conexión

python

def obtener\_modelos(marca\_seleccionada):

conn = conectar\_db()

try:

modelos\_df = pd.read\_sql\_query(

"SELECT DISTINCT modelo FROM catalogo WHERE marca = ? ORDER BY modelo ASC",

conn, params=(marca\_seleccionada,))

return modelos\_df["modelo"].tolist()

finally:

conn.close() # Asegúrate de cerrar la conexión

El uso de finally garantiza que la conexión se cierre incluso si ocurre un error.

**2. Evita accesos simultáneos**

Asegúrate de que no haya múltiples procesos o hilos intentando acceder a la base de datos al mismo tiempo. Si estás utilizando **Streamlit**, cada recarga o interacción puede abrir una nueva conexión.

Para manejar esto, usa un patrón de conexión compartida o asegúrate de abrir y cerrar conexiones dentro de cada función.

**3. Usa un modo de acceso diferente**

SQLite tiene un modo de acceso que impide bloqueos al permitir solo lecturas concurrentes. Cambia el URI de conexión así:

python

def conectar\_db():

conn = sqlite3.connect("file:checklist\_data\_center.db?mode=rw", uri=True)

return conn

Esto asegura que la conexión se realiza en modo **read-write** sin bloquear la base de datos.

**4. Verifica los procesos que están bloqueando la base de datos**

Si sospechas que algo externo está bloqueando la base de datos:

1. Usa una herramienta como **DB Browser for SQLite** para comprobar si hay conexiones abiertas.
2. Reinicia la aplicación de Streamlit para liberar cualquier proceso residual:

bash

Ctrl + C

streamlit run app.py

**5. Asegura el cierre de la base de datos al guardar**

Si hay consultas como INSERT o UPDATE, asegúrate de llamar a conn.commit() y conn.close() después de ejecutarlas.

Por ejemplo, en app.py:

python

if st.button("Guardar solicitud de ingreso"):

conn = conectar\_db()

try:

cursor = conn.cursor()

for equipo in equipos:

cursor.execute("""

INSERT INTO solicitudes\_ingreso (

solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

marca, modelo, numero\_serie, tipo\_equipo

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

equipo["marca"], equipo["modelo"], equipo["serie"], "Automático"))

conn.commit() # Asegúrate de guardar los cambios

finally:

conn.close() # Siempre cierra la conexión

st.success("¡Solicitud de ingreso guardada correctamente!")

**Resumen de ajustes**

1. **Revisa las funciones** obtener\_marcas **y** obtener\_modelos **para garantizar que las conexiones se cierren.**
2. **Evita accesos simultáneos abriendo y cerrando conexiones dentro de cada interacción.**
3. **Usa un patrón de conexión de solo lectura si el acceso es concurrente.**
4. **Confirma que todas las consultas de escritura (**INSERT**,** UPDATE**) finalicen con** commit() **y** close()**.**

Prueba estas correcciones y verifica si el problema desaparece. ¡Avísame si necesitas más orientación! 😊🚀

Ayer

ahora cargo, pero sigue con el problema que en una celda datacenter viene los datos de datacenter, ticket, marca y modelo y no por separado.

Entendido, Roberto. Este problema sucede cuando los valores de las columnas **datacenter**, **ticket**, **marca**, y **modelo** se están combinando en una sola celda de la tabla, en lugar de almacenarse separadamente en sus respectivas columnas. Esto generalmente ocurre si hay un error en el comando INSERT INTO o en cómo se están pasando los valores a la base de datos.

**Solución: Revisar el comando SQL y los valores que se están insertando**

**1. Verifica el comando SQL en el checklist de egreso**

El comando INSERT INTO debe especificar cada columna correctamente, con valores separados para cada campo. Aquí está el comando correcto que deberías estar utilizando:

python

cursor.execute("""

INSERT INTO solicitudes\_egreso (

solicitante, fecha\_egreso, datacenter, ticket, marca, modelo, numero\_serie,

rack\_origen, motivo\_egreso, estado\_salida

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_egreso, datacenter\_egreso, ticket\_egreso,

equipo["marca"], equipo["modelo"], equipo["serie"],

equipo["rack\_origen"], equipo["motivo\_egreso"], equipo["estado\_salida"]))

Este comando inserta cada valor en su columna correspondiente: **datacenter**, **ticket**, **marca**, y **modelo**, etc.

**2. Depura los valores antes de insertarlos**

Es posible que los valores que se están pasando al comando INSERT estén concatenados, causando que múltiples datos se guarden en una sola celda. Agrega un print para verificar los valores antes de la inserción:

python

if st.button("Guardar solicitud de egreso"):

print("Datos para guardar:")

print(datacenter\_egreso, ticket\_egreso, equipo["marca"], equipo["modelo"], equipo["serie"])

Esto te permitirá confirmar si los valores están correctamente separados.

**3. Validación:**

* **Verifica que** datacenter\_egreso **y** ticket\_egreso **sean valores únicos**.
* **Asegúrate de que** equipo["marca"] **y** equipo["modelo"] **no estén concatenados**.

Ejemplo de cómo los valores deberían aparecer:

python

# Ejemplo correcto:

datacenter\_egreso = "Providencia"

ticket\_egreso = "12345"

equipo["marca"] = "Dell"

equipo["modelo"] = "PowerEdge R740"

**4. Ajusta la estructura de la base de datos**

Verifica en tu tabla solicitudes\_egreso que las columnas estén definidas correctamente para aceptar valores separados.

La estructura debería verse así:

python

cursor.execute("""

CREATE TABLE IF NOT EXISTS solicitudes\_egreso (

id INTEGER PRIMARY KEY AUTOINCREMENT,

solicitante TEXT NOT NULL,

fecha\_egreso TEXT NOT NULL,

datacenter TEXT NOT NULL,

ticket TEXT NOT NULL,

marca TEXT NOT NULL,

modelo TEXT NOT NULL,

numero\_serie TEXT NOT NULL,

rack\_origen TEXT,

motivo\_egreso TEXT NOT NULL,

estado\_salida TEXT NOT NULL

)

""")

**Pasos para probar los ajustes**

1. **Actualiza el código en** app.py:
   * Usa el comando SQL correcto y verifica los valores antes de insertar.
2. **Ejecuta** database.py:
   * Asegúrate de que la tabla solicitudes\_egreso tenga columnas separadas para **datacenter**, **ticket**, **marca**, y **modelo**.
3. **Prueba el checklist de egreso**:
   * Completa los datos del formulario en Streamlit.
   * Verifica en la base de datos que los valores se guardan en las columnas correctas.

Si el problema persiste, comparte más detalles sobre cómo se están mostrando los datos en la base de datos y seguimos afinando juntos. ¡Vamos a resolverlo! 😊🚀

entregame el codigo final

¡Entendido, Roberto! Aquí tienes el **código final** de tu aplicación con todos los ajustes que hemos realizado. Esto incluye:

1. **Checklist de Ingreso**: Funcionalidades completas con orden alfabético, buscadores dinámicos, y almacenamiento correcto en la base de datos.
2. **Checklist de Egreso**: Incluye los ajustes para que los campos de **DataCenter**, **Ticket**, **Marca**, y **Modelo** se almacenen correctamente en columnas separadas.
3. Garantía de cierre de conexiones a la base de datos para evitar bloqueos.

**Código completo para app.py**

python

import streamlit as st

import sqlite3

import pandas as pd

# Función para conectar a la base de datos

def conectar\_db():

conn = sqlite3.connect("checklist\_data\_center.db")

return conn

# Función para obtener marcas únicas desde el catálogo

def obtener\_marcas():

conn = conectar\_db()

try:

marcas\_df = pd.read\_sql\_query("SELECT DISTINCT marca FROM catalogo ORDER BY marca ASC", conn)

return marcas\_df["marca"].tolist()

finally:

conn.close() # Asegúrate de cerrar la conexión

# Función para obtener modelos en función de la marca seleccionada

def obtener\_modelos(marca\_seleccionada):

conn = conectar\_db()

try:

modelos\_df = pd.read\_sql\_query(

"SELECT DISTINCT modelo FROM catalogo WHERE marca = ? ORDER BY modelo ASC",

conn, params=(marca\_seleccionada,))

return modelos\_df["modelo"].tolist()

finally:

conn.close() # Asegúrate de cerrar la conexión

# Función para obtener clientes

def obtener\_clientes():

return ["Cliente A", "Cliente B", "Cliente C"]

# Pantalla inicial para elegir el tipo de checklist

st.title("Checklist de Data Center")

tipo\_checklist = st.selectbox("Seleccione el tipo de checklist:", ["Ingreso", "Egreso", "Movimiento interno"])

# Lógica para el checklist de ingreso

if tipo\_checklist == "Ingreso":

st.subheader("Checklist de ingreso")

# Datos básicos del solicitante

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_ingreso = st.date\_input("Fecha de ingreso:")

cliente = st.selectbox("Seleccione el cliente:", obtener\_clientes())

datacenter = st.selectbox("Seleccione el DataCenter:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

ticket = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):")

# Cantidad de equipos a ingresar

cantidad\_equipos = st.number\_input("Cantidad de equipos a ingresar:", min\_value=1, step=1)

# Campos dinámicos para cada equipo

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

# Buscador de marcas

marca\_busqueda = st.text\_input(f"Buscar marca (Equipo {i}):", key=f"marca\_busqueda\_{i}")

marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()]

marca\_seleccionada = st.selectbox(f"Seleccione la marca del equipo {i}:", marcas, key=f"marca\_{i}")

# Buscador de modelos

modelo\_busqueda = st.text\_input(f"Buscar modelo (Equipo {i}):", key=f"modelo\_busqueda\_{i}")

modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()]

modelo\_seleccionado = st.selectbox(f"Seleccione el modelo del equipo {i}:", modelos, key=f"modelo\_{i}")

# Número de serie

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

equipos.append({"marca": marca\_seleccionada, "modelo": modelo\_seleccionado, "serie": numero\_serie})

# Botón para guardar la solicitud

if st.button("Guardar solicitud de ingreso"):

if all([solicitante, cliente, datacenter, ticket]) and all(e["serie"] for e in equipos):

conn = conectar\_db()

try:

cursor = conn.cursor()

for equipo in equipos:

cursor.execute("""

INSERT INTO solicitudes\_ingreso (

solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

marca, modelo, numero\_serie, tipo\_equipo

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

equipo["marca"], equipo["modelo"], equipo["serie"], "Automático"))

conn.commit()

st.success("¡Solicitud de ingreso guardada correctamente!")

finally:

conn.close() # Siempre cierra la conexión

else:

st.error("Por favor, complete todos los campos obligatorios.")

# Lógica para el checklist de egreso

elif tipo\_checklist == "Egreso":

st.subheader("Checklist de egreso")

# Datos básicos del solicitante

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_egreso = st.date\_input("Fecha de egreso:")

datacenter\_egreso = st.selectbox("Seleccione el DataCenter de egreso:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

ticket\_egreso = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):")

# Cantidad de equipos a egresar

cantidad\_equipos = st.number\_input("Cantidad de equipos a egresar:", min\_value=1, step=1)

# Campos dinámicos para cada equipo

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

# Buscador de marcas

marca\_busqueda = st.text\_input(f"Buscar marca (Equipo {i}):", key=f"marca\_busqueda\_{i}")

marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()]

marca\_seleccionada = st.selectbox(f"Seleccione la marca del equipo {i}:", marcas, key=f"marca\_{i}")

# Buscador de modelos

modelo\_busqueda = st.text\_input(f"Buscar modelo (Equipo {i}):", key=f"modelo\_busqueda\_{i}")

modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()]

modelo\_seleccionado = st.selectbox(f"Seleccione el modelo del equipo {i}:", modelos, key=f"modelo\_{i}")

# Número de serie

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

# Rack de origen

rack\_origen = st.text\_input(f"Rack de origen del equipo {i}:", key=f"rack\_{i}")

# Motivo del egreso

motivo\_egreso = st.selectbox(f"Motivo del egreso (Equipo {i}):", ["Desinstalación", "Traslado", "Otro"], key=f"motivo\_{i}")

# Estado del equipo

estado\_salida = st.selectbox(f"Estado del equipo al salir (Equipo {i}):", ["Funcional", "Dañado"], key=f"estado\_{i}")

equipos.append({

"marca": marca\_seleccionada,

"modelo": modelo\_seleccionado,

"serie": numero\_serie,

"rack\_origen": rack\_origen,

"motivo\_egreso": motivo\_egreso,

"estado\_salida": estado\_salida

})

# Botón para guardar la solicitud

if st.button("Guardar solicitud de egreso"):

if all([solicitante, fecha\_egreso, datacenter\_egreso, ticket\_egreso]) and all(e["serie"] for e in equipos):

conn = conectar\_db()

try:

cursor = conn.cursor()

for equipo in equipos:

cursor.execute("""

INSERT INTO solicitudes\_egreso (

solicitante, fecha\_egreso, datacenter, ticket, marca, modelo, numero\_serie,

rack\_origen, motivo\_egreso, estado\_salida

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_egreso, datacenter\_egreso, ticket\_egreso,

equipo["marca"], equipo["modelo"], equipo["serie"],

equipo["rack\_origen"], equipo["motivo\_egreso"], equipo["estado\_salida"]))

conn.commit()

st.success("¡Solicitud de egreso guardada correctamente!")

finally:

conn.close() # Siempre cierra la conexión

else:

st.error("Por favor, complete todos los campos obligatorios.")

# Espacio para lógica de movimiento interno (por implementar)

elif tipo\_checklist == "Movimiento interno":

st.subheader("Checklist de movimiento interno")

st.write("Esta sección aún está en desarrollo.")

**Prueba final**

1. Guarda este código en tu archivo app.py.
2. Ejecuta en tu terminal:

bash

streamlit run app.py

1. Verifica que:
   * Los datos del checklist de ingreso y egreso se guarden

me sigue apareciendo en una celda esos 4 datos

¡Entendido, Roberto! Esto indica que, aunque el código parece estar correctamente configurado, hay algo en cómo los datos están siendo pasados o insertados en la tabla que está concatenando los valores en una sola celda. Vamos a resolver esto paso a paso:

**1. Revisemos el comando SQL**

El problema más probable radica en el comando INSERT INTO. Debemos garantizar que cada valor se inserte en su columna correspondiente de manera separada.

Aquí está el comando que **debe estar en tu código** para el checklist de egreso:

python

cursor.execute("""

INSERT INTO solicitudes\_egreso (

solicitante, fecha\_egreso, datacenter, ticket, marca, modelo, numero\_serie,

rack\_origen, motivo\_egreso, estado\_salida

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_egreso, datacenter\_egreso, ticket\_egreso,

equipo["marca"], equipo["modelo"], equipo["serie"],

equipo["rack\_origen"], equipo["motivo\_egreso"], equipo["estado\_salida"]))

Verifica que:

1. **Los valores están correctamente separados.**
2. **Las columnas** datacenter**,** ticket**,** marca**, y** modelo **no están siendo concatenadas antes de pasar al comando SQL.**

**2. Depura los valores antes de la inserción**

Agrega un print justo antes del INSERT INTO para verificar qué valores están siendo enviados al comando. Esto te ayudará a identificar si los datos ya vienen concatenados desde el formulario.

Por ejemplo, en el bloque de egreso:

python

if st.button("Guardar solicitud de egreso"):

for equipo in equipos:

# Imprime los valores antes de insertarlos en la base de datos

print("Valores enviados al comando SQL:")

print(datacenter\_egreso, ticket\_egreso, equipo["marca"], equipo["modelo"], equipo["serie"])

cursor.execute("""

INSERT INTO solicitudes\_egreso (

solicitante, fecha\_egreso, datacenter, ticket, marca, modelo, numero\_serie,

rack\_origen, motivo\_egreso, estado\_salida

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_egreso, datacenter\_egreso, ticket\_egreso,

equipo["marca"], equipo["modelo"], equipo["serie"],

equipo["rack\_origen"], equipo["motivo\_egreso"], equipo["estado\_salida"]))

Revisa el **output** en la terminal para confirmar que los valores están correctamente separados.

**3. Verifica la estructura de la tabla**

Asegúrate de que la tabla solicitudes\_egreso tiene las columnas separadas y que no están siendo interpretadas como un único campo.

Estructura correcta de la tabla en database.py:

python

cursor.execute("DROP TABLE IF EXISTS solicitudes\_egreso")

cursor.execute("""

CREATE TABLE IF NOT EXISTS solicitudes\_egreso (

id INTEGER PRIMARY KEY AUTOINCREMENT,

solicitante TEXT NOT NULL,

fecha\_egreso TEXT NOT NULL,

datacenter TEXT NOT NULL,

ticket TEXT NOT NULL,

marca TEXT NOT NULL,

modelo TEXT NOT NULL,

numero\_serie TEXT NOT NULL,

rack\_origen TEXT,

motivo\_egreso TEXT NOT NULL,

estado\_salida TEXT NOT NULL

)

""")

Ejecuta database.py nuevamente para asegurarte de que la estructura esté correctamente configurada.

**4. Problema con concatenación en el formulario**

Es posible que los valores se estén concatenando en el momento en que los datos se recogen del formulario. Por ejemplo, si los inputs de datacenter\_egreso, ticket\_egreso, equipo["marca"], y equipo["modelo"] están siendo combinados accidentalmente.

Asegúrate de que cada campo del formulario esté definiendo valores únicos:

python

datacenter\_egreso = st.selectbox("Seleccione el DataCenter de egreso:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

ticket\_egreso = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):")

marca\_busqueda = st.text\_input("Buscar marca (Equipo):")

modelo\_busqueda = st.text\_input("Buscar modelo (Equipo):")

**5. Limpia los datos en la base de datos**

Si el problema persiste, hay que revisar directamente la base de datos para identificar cómo los valores están siendo almacenados.

Usa **DB Browser for SQLite** para ejecutar una consulta y verificar los registros en la tabla solicitudes\_egreso:

sql

SELECT \* FROM solicitudes\_egreso;

Esto te permitirá inspeccionar cómo los datos están siendo almacenados y si los valores están correctamente separados.

**Prueba y confirmación**

1. Revisa los valores enviados con el print.
2. Asegúrate de que los valores del formulario están correctamente definidos y separados.
3. Verifica la estructura de la tabla en la base de datos.

Si el problema persiste después de estos ajustes, envíame detalles adicionales (como los resultados del print) y lo resolvemos juntos. ¡Estamos cerca! 🚀😊

al revisar la base de datos DB Browser for Sqlite, este me muestra la informacion correctasmente en sus celdas, pero al verlas en Visual code, me aparece esas cuatro columnas en 1 sola

Eso confirma que los datos están correctamente almacenados en tu base de datos, pero el problema radica en cómo Visual Studio Code está presentando la información. Esto generalmente ocurre si el resultado de la consulta a la base de datos (en tu código o en alguna visualización) se está procesando de manera que concatena las columnas en una sola cadena.

**1. Revisa cómo estás mostrando los datos en Visual Studio Code**

Es posible que el problema provenga de cómo estás generando o representando los datos en tu aplicación. Si estás usando una consulta como pd.read\_sql\_query, es probable que haya un error en el procesamiento o en cómo se están manipulando las columnas.

Por ejemplo:

python

marcas\_df = pd.read\_sql\_query("SELECT DISTINCT marca FROM catalogo ORDER BY marca ASC", conn)

Este comando debería estar devolviendo datos correctamente separados. Si ves datos concatenados, verifica cómo los estás usando o imprimiendo.

**2. Asegúrate de que cada columna se está utilizando de forma independiente**

Cuando trabajes con el resultado de la consulta, evita concatenar columnas en una sola cadena. Por ejemplo, si estás mostrando los datos como una lista o imprimiéndolos:

python

for index, row in marcas\_df.iterrows():

print(row["marca"])

Esto asegura que cada columna sea independiente al imprimirla o procesarla.

**3. Configuración específica de Visual Studio Code**

Si estás visualizando los datos desde un archivo exportado (como un CSV o JSON), podría haber un problema en el formato del archivo. Revisa cómo estás exportando los datos, ya sea usando to\_csv() o alguna otra función.

Por ejemplo, usa algo como esto para exportar correctamente las columnas separadas:

python

marcas\_df.to\_csv("output.csv", index=False)

Luego abre ese archivo en Visual Studio Code para confirmar que las columnas están separadas correctamente.

**4. Depuración adicional en el código**

Si estás viendo los datos desde un DataFrame en Pandas en Visual Studio Code, imprime los datos directamente antes de exportarlos o procesarlos, para asegurarte de que no están concatenados:

python

print(marcas\_df.head())

**5. Herramientas alternativas para confirmar la separación**

Si el problema persiste en Visual Studio Code, puedes usar herramientas como Excel o directamente el terminal para abrir el archivo de salida y confirmar que los datos están correctamente separados.

**Siguientes pasos**

Confirma cómo estás manipulando o mostrando los datos en tu aplicación. Si necesitas que revise una parte específica del código en Visual Studio Code, envíame más detalles y seguimos ajustando. ¡Vamos a resolver esto! 😊🚀

5. Herramientas alternativas para confirmar la separación Si el problema persiste en Visual Studio Code, puedes usar herramientas como Excel o directamente el terminal para abrir el archivo de salida y confirmar que los datos están correctamente separados. ayudame en este paso

Claro que sí, Roberto. Aquí te detallo los pasos para usar herramientas alternativas como **Excel** y el terminal para confirmar que los datos de tu base de datos están correctamente separados:

**Usando Excel para confirmar la separación de columnas**

1. **Exporta los datos de tu base de datos a un archivo CSV:**
   * Si estás usando Python, puedes exportar los datos desde un DataFrame con:

python

marcas\_df.to\_csv("output.csv", index=False)

* + Esto creará un archivo llamado output.csv en la misma carpeta donde estás trabajando.

1. **Abre el archivo CSV en Excel:**
   * Ve al explorador de archivos y localiza el archivo output.csv.
   * Haz clic derecho sobre él y selecciona **Abrir con > Excel**.
   * En Excel, verifica que los datos estén organizados correctamente en columnas separadas.
2. **Opcional: Usar el asistente de texto a columnas (si las columnas están unidas):**
   * Si al abrir el archivo en Excel ves datos concatenados en una celda, selecciona la columna afectada.
   * Ve a la pestaña **Datos** > **Texto en columnas**.
   * Selecciona **Delimitado** y presiona **Siguiente**.
   * Marca el delimitador adecuado, como coma , o tabulación (según cómo se exportaron los datos).
   * Presiona **Finalizar** para separar los datos en columnas.

**Usando el terminal para confirmar la separación**

1. **Abre el archivo CSV en el terminal o en un editor de texto:**
   * Usa el comando cat (Linux/macOS) o type (Windows) para mostrar el contenido del archivo:

bash

cat output.csv

(En Windows, usa type output.csv).

* + Esto mostrará el contenido del archivo directamente en el terminal, con los valores separados por comas (,) si el archivo fue exportado correctamente.

1. **Revisar datos separados por delimitadores:**
   * Asegúrate de que cada columna esté separada por una coma (o el delimitador especificado durante la exportación).
2. **Usa un programa como** less **o** more **para revisar el archivo línea por línea:**
   * Esto es útil si el archivo es grande:

bash

less output.csv

* + Navega con las flechas hacia arriba y abajo para explorar el archivo.

**Prueba directa desde Python**

Si exportaste el archivo desde Python pero quieres verificar los datos directamente en tu código, puedes cargar el archivo CSV nuevamente en un DataFrame y mostrar las primeras filas:

python

df = pd.read\_csv("output.csv")

print(df.head())

Esto te mostrará cómo se ven las columnas y los valores en Python, similar a cómo se procesarían en Visual Studio Code.

**Qué hacer si persiste el problema**

* Si al abrir en Excel o en el terminal los datos están bien separados, pero en Visual Studio Code continúan concatenados, podría ser un problema de formato o de cómo Visual Studio Code está interpretando los datos.
  + Asegúrate de que estás abriendo el archivo con el formato adecuado (por ejemplo, como CSV y no como un archivo de texto plano).

¡Dime si necesitas más ayuda con estos pasos o con alguna herramienta específica! 😊🚀

Usando Excel para confirmar la separación de columnas, ayudame porfavor en esta parte

¡Por supuesto, Roberto! Aquí tienes una guía clara para usar Excel y confirmar que las columnas están correctamente separadas en tu archivo CSV:

**Paso 1: Localiza tu archivo CSV**

* Asegúrate de haber exportado los datos desde tu aplicación o base de datos en un archivo CSV llamado, por ejemplo, output.csv.
* Este archivo debe estar en la carpeta de tu proyecto o en una ubicación que puedas encontrar fácilmente.

**Paso 2: Abre el archivo en Excel**

1. **Encuentra el archivo:**
   * Ve a tu explorador de archivos (Windows: File Explorer, macOS: Finder).
   * Busca el archivo output.csv.
2. **Abrir con Excel:**
   * Haz clic derecho sobre el archivo y selecciona **Abrir con** > **Excel**.
   * Alternativamente, si Excel es tu programa predeterminado para abrir archivos CSV, simplemente haz doble clic en el archivo.

**Paso 3: Verifica la separación de columnas**

1. Una vez abierto el archivo en Excel:
   * Los datos deberían aparecer organizados en columnas separadas, como:
   * | Datacenter | Ticket | Marca | Modelo |
   * | Providencia | 12345 | Dell | PowerEdge R740 |
2. Si los datos están concatenados en una sola celda:
   * Selecciona la columna afectada (la que contiene todos los datos combinados).
   * Ve a la pestaña **Datos** en el menú de Excel.
   * Haz clic en **Texto en columnas**.

**Paso 4: Usa la herramienta "Texto en columnas"**

1. **Selecciona "Delimitado":**
   * Cuando se abra el asistente, selecciona **Delimitado** como el tipo de datos.
   * Presiona **Siguiente**.
2. **Selecciona el delimitador adecuado:**
   * Si exportaste los datos con comas (,) como separador, marca la opción **Coma**.
   * Si exportaste los datos con tabulación como separador, marca la opción **Tabulación**.
3. **Verifica la vista previa:**
   * En la parte inferior del asistente, verás una vista previa de cómo se dividirán los datos en columnas.
4. **Finaliza el proceso:**
   * Presiona **Finalizar** para aplicar los cambios.
   * Los datos ahora deberían estar correctamente organizados en columnas separadas.

**Paso 5: Guarda el archivo**

1. Una vez que los datos estén separados correctamente:
   * Ve a **Archivo** > **Guardar como**.
   * Guarda el archivo con un nombre como output\_correcto.csv para identificarlo fácilmente.

**¿Qué hacer si persisten los problemas?**

Si los datos siguen concatenados o no se separan correctamente:

* Asegúrate de que el archivo CSV esté utilizando comas o tabulaciones como delimitadores.
* Puedes abrir el archivo en un editor de texto (como Notepad en Windows o TextEdit en macOS) y verificar manualmente el formato.

¡Déjame saber cómo te va o si necesitas más ayuda con alguno de estos pasos! 😊🚀

donde coloco el coduigo del dataframe que me indicaste?

El código del DataFrame que te indiqué es útil si quieres inspeccionar o trabajar directamente con los datos dentro de tu script de Python. Puedes colocarlo en cualquier lugar donde necesites verificar los datos de tu base de datos o de un archivo CSV. Por ejemplo, si exportaste los datos a un archivo CSV (output.csv), este código puede ser colocado en tu archivo Python para cargar y verificar la estructura y el contenido del archivo.

**Ejemplo de uso del DataFrame para inspección**

Si estás trabajando en tu archivo app.py, coloca este código después de haber exportado el archivo CSV o en una sección donde estés analizando los datos. Aquí tienes un ejemplo:

python

import pandas as pd

# Cargar el archivo CSV generado

df = pd.read\_csv("output.csv")

# Inspeccionar las primeras filas del archivo para verificar que las columnas estén separadas correctamente

print(df.head())

# Opcional: mostrar información adicional sobre el DataFrame

print("Columnas del archivo:")

print(df.columns)

# Si deseas verificar algún dato en particular

for index, row in df.iterrows():

print(f"Datacenter: {row['Datacenter']}, Ticket: {row['Ticket']}, Marca: {row['Marca']}, Modelo: {row['Modelo']}")

**Cómo usarlo en tu flujo**

1. **Exporta el archivo CSV desde tu base de datos**:
   * Usa to\_csv() para guardar los datos en un archivo CSV, como output.csv.
2. **Coloca este código en tu script Python**:
   * Después de exportar o generar el archivo CSV, utiliza este código para cargar el archivo y verificar que las columnas y datos estén correctos.
3. **Ejecuta el archivo Python**:
   * Abre tu terminal y corre el archivo para confirmar que los datos están organizados correctamente:

bash

python app.py

1. **Verifica en la salida de la terminal**:
   * Mira la impresión de df.head() para confirmar que los datos están separados en columnas independientes.

**Prueba en Visual Studio Code**

Si estás trabajando con datos desde el archivo, puedes usar este fragmento dentro de tu flujo de trabajo en Visual Studio Code y ejecutar tu script para visualizar la salida en la terminal.

¿Te gustaría ayuda para agregar este fragmento en un lugar específico de tu código? 😊🚀

Usando Excel para confirmar la separación de columnas Exporta los datos de tu base de datos a un archivo CSV: Si estás usando Python, puedes exportar los datos desde un DataFrame con: python marcas\_df.to\_csv("output.csv", index=False) Esto creará un archivo llamado output.csv en la misma carpeta donde estás trabajando. Abre el archivo CSV en Excel: Ve al explorador de archivos y localiza el archivo output.csv. Haz clic derecho sobre él y selecciona Abrir con > Excel. En Excel, verifica que los datos estén organizados correctamente en columnas separadas. Opcional: Usar el asistente de texto a columnas (si las columnas están unidas): Si al abrir el archivo en Excel ves datos concatenados en una celda, selecciona la columna afectada. Ve a la pestaña Datos > Texto en columnas. Selecciona Delimitado y presiona Siguiente. Marca el delimitador adecuado, como coma , o tabulación (según cómo se exportaron los datos). Presiona Finalizar para separar los datos en columnas. no entendi nada de esto, donde coloco el codigo, y como lo hago?

¡Claro, Roberto! Aquí tienes una guía detallada paso a paso para entender cómo exportar tus datos a un archivo CSV y luego abrirlos en Excel:

**Parte 1: Exportar los datos a un archivo CSV**

Para exportar los datos de tu base de datos a un archivo CSV utilizando Python, sigue estos pasos:

1. **Ubicación del código:**
   * Coloca el código para exportar el CSV en tu archivo Python donde estés manejando los datos (por ejemplo, app.py).
2. **Ejemplo de código para exportar:** Si tienes un DataFrame marcas\_df, puedes usar este código:

python

import pandas as pd

import sqlite3

# Conectar a la base de datos

conn = sqlite3.connect("checklist\_data\_center.db")

# Consulta para obtener datos

marcas\_df = pd.read\_sql\_query("SELECT \* FROM catalogo", conn)

# Exportar a CSV

marcas\_df.to\_csv("output.csv", index=False)

print("Archivo CSV exportado correctamente como output.csv.")

conn.close()

1. **Qué hace este código:**
   * Crea una conexión con tu base de datos SQLite.
   * Ejecuta una consulta SQL para obtener los datos de la tabla catalogo.
   * Exporta los datos a un archivo CSV llamado output.csv en la misma carpeta donde está el script.
2. **Cómo ejecutarlo:**
   * Guarda este código en tu archivo Python (por ejemplo, export\_csv.py).
   * En tu terminal, ejecuta:

bash

python export\_csv.py

1. **Confirma que el archivo CSV se ha creado:**
   * Ve a tu carpeta de proyecto y busca el archivo llamado output.csv.

**Parte 2: Abrir el archivo CSV en Excel**

Ahora que has exportado el archivo CSV, vamos a abrirlo en Excel:

1. **Localiza el archivo:**
   * Ve a tu explorador de archivos y encuentra el archivo output.csv.
2. **Abrir con Excel:**
   * Haz clic derecho en el archivo output.csv.
   * Selecciona **Abrir con** > **Microsoft Excel**.
3. **Verifica los datos:**
   * Una vez abierto en Excel, los datos deberían aparecer organizados en columnas separadas.

**Parte 3: Separar columnas en Excel si los datos están unidos**

Si al abrir el archivo en Excel ves que los datos están todos concatenados en una sola celda, sigue estos pasos:

1. **Selecciona la columna afectada:**
   * Haz clic en la columna que contiene los datos concatenados.
2. **Usa la herramienta "Texto en columnas":**
   * Ve a la pestaña **Datos** en el menú de Excel.
   * Haz clic en **Texto en columnas**.
3. **Configura el asistente:**
   * Selecciona **Delimitado** como tipo de datos.
   * Presiona **Siguiente**.
   * Marca el delimitador adecuado:
     + **Coma (,)** si el archivo CSV usa comas como separador.
     + **Tabulación** si usa tabulaciones.
4. **Finaliza la separación:**
   * Presiona **Finalizar** para dividir los datos en columnas separadas.

**Qué hacer después**

1. Si los datos ahora están separados correctamente en Excel, puedes guardar el archivo modificado.
   * Ve a **Archivo** > **Guardar como** y guarda el archivo con otro nombre para no sobrescribir el original, por ejemplo: output\_correcto.xlsx.
2. Si tienes dudas al realizar cualquiera de estos pasos, avísame y puedo explicarte cada uno en más detalle.

Espero que esto lo haga más claro para ti. ¡Avísame cómo te va! 😊🚀

si con excel se puede ver correectamente el llenado, es problema del visual code. contnuemos con el proyecto

Perfecto, Roberto. ¡Qué bueno que ya verificamos que los datos están bien en la base de datos y en Excel! Si Visual Studio Code no interpreta correctamente los datos, probablemente es solo un tema de visualización y no afecta la integridad del proyecto. Continuemos con el siguiente punto para seguir avanzando en tu proyecto.

**Opciones para continuar:**

1. **Checklist de Movimiento Interno:** Podríamos implementar un formulario para registrar cambios o reubicaciones de equipos dentro del Data Center.
2. **Interfaz para gestión adicional:** Crear vistas o herramientas para buscar, filtrar o actualizar registros en la base de datos.
3. **Generación de reportes:** Automatizar la creación de reportes (por ejemplo, en Excel o PDF) basados en los datos registrados.
4. **Mejoras en funcionalidades actuales:** Si hay algo más en los checklists de ingreso o egreso que te gustaría refinar.

¿En qué punto te gustaría enfocarte ahora? ¡Vamos con todo, Roberto! 😊🚀

checklist de movimiento interno, apliquemos lo mismo que los anteriores, con los buscadores, filtros etc, agregado ticket y todo lo necesario.

¡Entendido, Roberto! Vamos a implementar el **Checklist de Movimiento Interno** con todas las funcionalidades avanzadas que hemos aplicado a los checklists anteriores (buscadores, filtros, validaciones, y campos necesarios como el Ticket). Este checklist permitirá registrar la transferencia de equipos dentro del Data Center, asegurando que toda la información esté correctamente registrada.

**Estructura del Checklist de Movimiento Interno**

Los datos requeridos serán:

1. **Información del solicitante**:
   * Nombre del solicitante.
   * Fecha del movimiento.
   * Ticket asociado.
2. **Datos del movimiento**:
   * DataCenter de origen.
   * Rack y ubicación en el DataCenter de origen.
   * DataCenter de destino.
   * Rack y ubicación en el DataCenter de destino.
3. **Información del equipo**:
   * Marca.
   * Modelo.
   * Número de serie.
   * Estado del equipo antes del movimiento.

**Código para el Checklist de Movimiento Interno**

Agrega este bloque de código en tu archivo app.py para implementar el checklist:

python

elif tipo\_checklist == "Movimiento interno":

st.subheader("Checklist de movimiento interno")

# Datos básicos del solicitante

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_movimiento = st.date\_input("Fecha del movimiento:")

ticket\_movimiento = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):")

# DataCenters de origen y destino

datacenter\_origen = st.selectbox("Seleccione el DataCenter de origen:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

rack\_origen = st.text\_input("Rack y ubicación del equipo en el DataCenter de origen:")

datacenter\_destino = st.selectbox("Seleccione el DataCenter de destino:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

rack\_destino = st.text\_input("Rack y ubicación del equipo en el DataCenter de destino:")

# Cantidad de equipos a mover

cantidad\_equipos = st.number\_input("Cantidad de equipos a mover:", min\_value=1, step=1)

# Campos dinámicos para cada equipo

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

# Buscador de marcas

marca\_busqueda = st.text\_input(f"Buscar marca (Equipo {i}):", key=f"marca\_busqueda\_{i}")

marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()]

marca\_seleccionada = st.selectbox(f"Seleccione la marca del equipo {i}:", marcas, key=f"marca\_{i}")

# Buscador de modelos

modelo\_busqueda = st.text\_input(f"Buscar modelo (Equipo {i}):", key=f"modelo\_busqueda\_{i}")

modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()]

modelo\_seleccionado = st.selectbox(f"Seleccione el modelo del equipo {i}:", modelos, key=f"modelo\_{i}")

# Número de serie

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

# Estado del equipo antes del movimiento

estado\_equipo = st.selectbox(f"Estado del equipo antes del movimiento (Equipo {i}):", ["Funcional", "Dañado"], key=f"estado\_equipo\_{i}")

equipos.append({

"marca": marca\_seleccionada,

"modelo": modelo\_seleccionado,

"serie": numero\_serie,

"estado\_equipo": estado\_equipo

})

# Botón para guardar la solicitud

if st.button("Guardar solicitud de movimiento interno"):

if all([solicitante, fecha\_movimiento, ticket\_movimiento, datacenter\_origen, rack\_origen, datacenter\_destino, rack\_destino]) and all(e["serie"] for e in equipos):

conn = conectar\_db()

try:

cursor = conn.cursor()

for equipo in equipos:

cursor.execute("""

INSERT INTO movimientos\_internos (

solicitante, fecha\_movimiento, ticket, datacenter\_origen, rack\_origen,

datacenter\_destino, rack\_destino, marca, modelo, numero\_serie, estado\_equipo

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_movimiento, ticket\_movimiento, datacenter\_origen, rack\_origen,

datacenter\_destino, rack\_destino, equipo["marca"], equipo["modelo"], equipo["serie"], equipo["estado\_equipo"]))

conn.commit()

st.success("¡Solicitud de movimiento interno guardada correctamente!")

finally:

conn.close() # Siempre cierra la conexión

else:

st.error("Por favor, complete todos los campos obligatorios.")

**Paso adicional: Crear la tabla en la base de datos**

Actualiza tu archivo database.py para crear la tabla movimientos\_internos. Agrega este código:

python

cursor.execute("DROP TABLE IF EXISTS movimientos\_internos")

cursor.execute("""

CREATE TABLE IF NOT EXISTS movimientos\_internos (

id INTEGER PRIMARY KEY AUTOINCREMENT,

solicitante TEXT NOT NULL,

fecha\_movimiento TEXT NOT NULL,

ticket TEXT NOT NULL,

datacenter\_origen TEXT NOT NULL,

rack\_origen TEXT,

datacenter\_destino TEXT NOT NULL,

rack\_destino TEXT,

marca TEXT NOT NULL,

modelo TEXT NOT NULL,

numero\_serie TEXT NOT NULL,

estado\_equipo TEXT NOT NULL

)

""")

Ejecuta database.py para asegurarte de que la tabla se crea correctamente:

bash

python database.py

**Prueba del Checklist de Movimiento Interno**

1. **Ejecuta la aplicación:**

bash

streamlit run app.py

1. **Selecciona la opción "Movimiento interno" y completa el formulario:**
   * Ingresa los datos del solicitante, DataCenters, racks, y equipos.
2. **Guarda la solicitud:**
   * Asegúrate de que no falten campos y de que todos los equipos estén registrados correctamente.
3. **Verifica en la base de datos:**
   * Usa **DB Browser for SQLite** para abrir la tabla movimientos\_internos y confirmar que los datos se guardaron correctamente.

**Características implementadas**

* **Buscadores de marcas y modelos**: Evita la duplicación y permite encontrar datos fácilmente.
* **Campos dinámicos por equipo**: Para manejar múltiples equipos en un solo formulario.
* **Ticket y estado del equipo**: Registro detallado de todos los datos relevantes del movimiento.
* **Validaciones**: Para garantizar que todos los campos obligatorios estén completos.

¿Quieres ajustar algo o seguimos perfeccionando el proyecto? ¡Vamos avanzando, Roberto! 😊🚀

me darias el codigo final porfavor

¡Claro, Roberto! Aquí tienes el **código final completo** con el Checklist de Movimiento Interno, además de los Checklists de Ingreso y Egreso, listos con todas las funcionalidades avanzadas que hemos aplicado.

**Código final para app.py**

python

import streamlit as st

import sqlite3

import pandas as pd

# Función para conectar a la base de datos

def conectar\_db():

conn = sqlite3.connect("checklist\_data\_center.db")

return conn

# Función para obtener marcas únicas desde el catálogo

def obtener\_marcas():

conn = conectar\_db()

try:

marcas\_df = pd.read\_sql\_query("SELECT DISTINCT marca FROM catalogo ORDER BY marca ASC", conn)

return marcas\_df["marca"].tolist()

finally:

conn.close()

# Función para obtener modelos en función de la marca seleccionada

def obtener\_modelos(marca\_seleccionada):

conn = conectar\_db()

try:

modelos\_df = pd.read\_sql\_query(

"SELECT DISTINCT modelo FROM catalogo WHERE marca = ? ORDER BY modelo ASC",

conn, params=(marca\_seleccionada,))

return modelos\_df["modelo"].tolist()

finally:

conn.close()

# Función para obtener clientes

def obtener\_clientes():

return ["Cliente A", "Cliente B", "Cliente C"]

# Pantalla inicial para elegir el tipo de checklist

st.title("Checklist de Data Center")

tipo\_checklist = st.selectbox("Seleccione el tipo de checklist:", ["Ingreso", "Egreso", "Movimiento interno"])

# Lógica para el checklist de ingreso

if tipo\_checklist == "Ingreso":

st.subheader("Checklist de ingreso")

# Datos básicos del solicitante

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_ingreso = st.date\_input("Fecha de ingreso:")

cliente = st.selectbox("Seleccione el cliente:", obtener\_clientes())

datacenter = st.selectbox("Seleccione el DataCenter:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

ticket = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):")

# Cantidad de equipos a ingresar

cantidad\_equipos = st.number\_input("Cantidad de equipos a ingresar:", min\_value=1, step=1)

# Campos dinámicos para cada equipo

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

# Buscador de marcas

marca\_busqueda = st.text\_input(f"Buscar marca (Equipo {i}):", key=f"marca\_busqueda\_{i}")

marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()]

marca\_seleccionada = st.selectbox(f"Seleccione la marca del equipo {i}:", marcas, key=f"marca\_{i}")

# Buscador de modelos

modelo\_busqueda = st.text\_input(f"Buscar modelo (Equipo {i}):", key=f"modelo\_busqueda\_{i}")

modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()]

modelo\_seleccionado = st.selectbox(f"Seleccione el modelo del equipo {i}:", modelos, key=f"modelo\_{i}")

# Número de serie

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

equipos.append({"marca": marca\_seleccionada, "modelo": modelo\_seleccionado, "serie": numero\_serie})

# Botón para guardar la solicitud

if st.button("Guardar solicitud de ingreso"):

if all([solicitante, cliente, datacenter, ticket]) and all(e["serie"] for e in equipos):

conn = conectar\_db()

try:

cursor = conn.cursor()

for equipo in equipos:

cursor.execute("""

INSERT INTO solicitudes\_ingreso (

solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

marca, modelo, numero\_serie, tipo\_equipo

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

equipo["marca"], equipo["modelo"], equipo["serie"], "Automático"))

conn.commit()

st.success("¡Solicitud de ingreso guardada correctamente!")

finally:

conn.close()

else:

st.error("Por favor, complete todos los campos obligatorios.")

# Lógica para el checklist de egreso

elif tipo\_checklist == "Egreso":

st.subheader("Checklist de egreso")

# Datos básicos del solicitante

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_egreso = st.date\_input("Fecha de egreso:")

datacenter\_egreso = st.selectbox("Seleccione el DataCenter de egreso:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

ticket\_egreso = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):")

# Cantidad de equipos a egresar

cantidad\_equipos = st.number\_input("Cantidad de equipos a egresar:", min\_value=1, step=1)

# Campos dinámicos para cada equipo

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

# Buscador de marcas

marca\_busqueda = st.text\_input(f"Buscar marca (Equipo {i}):", key=f"marca\_busqueda\_{i}")

marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()]

marca\_seleccionada = st.selectbox(f"Seleccione la marca del equipo {i}:", marcas, key=f"marca\_{i}")

# Buscador de modelos

modelo\_busqueda = st.text\_input(f"Buscar modelo (Equipo {i}):", key=f"modelo\_busqueda\_{i}")

modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()]

modelo\_seleccionado = st.selectbox(f"Seleccione el modelo del equipo {i}:", modelos, key=f"modelo\_{i}")

# Número de serie

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

# Rack de origen

rack\_origen = st.text\_input(f"Rack de origen del equipo {i}:", key=f"rack\_{i}")

# Motivo del egreso

motivo\_egreso = st.selectbox(f"Motivo del egreso (Equipo {i}):", ["Desinstalación", "Traslado", "Otro"], key=f"motivo\_{i}")

# Estado del equipo

estado\_salida = st.selectbox(f"Estado del equipo al salir (Equipo {i}):", ["Funcional", "Dañado"], key=f"estado\_{i}")

equipos.append({

"marca": marca\_seleccionada,

"modelo": modelo\_seleccionado,

"serie": numero\_serie,

"rack\_origen": rack\_origen,

"motivo\_egreso": motivo\_egreso,

"estado\_salida": estado\_salida

})

# Botón para guardar la solicitud

if st.button("Guardar solicitud de egreso"):

if all([solicitante, fecha\_egreso, datacenter\_egreso, ticket\_egreso]) and all(e["serie"] for e in equipos):

conn = conectar\_db()

try:

cursor = conn.cursor()

for equipo in equipos:

cursor.execute("""

INSERT INTO solicitudes\_egreso (

solicitante, fecha\_egreso, datacenter, ticket, marca, modelo, numero\_serie,

rack\_origen, motivo\_egreso, estado\_salida

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_egreso, datacenter\_egreso, ticket\_egreso,

equipo["marca"], equipo["modelo"], equipo["serie"],

equipo["rack\_origen"], equipo["motivo\_egreso"], equipo["estado\_salida"]))

conn.commit()

st.success("¡Solicitud de egreso guardada correctamente!")

finally:

conn.close()

else:

st.error("Por favor, complete todos los campos obligatorios.")

# Lógica para el checklist de movimiento interno

elif tipo\_checklist == "Movimiento interno":

st.subheader("Checklist de movimiento interno")

# Datos básicos del solicitante

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_movimiento = st.date\_input("Fecha del movimiento:")

ticket\_movimiento = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):")

# DataCenters de origen y destino

datacenter\_origen = st.selectbox("Seleccione el DataCenter de origen:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

rack\_origen = st.text\_input("Rack y ubicación del equipo en el DataCenter de origen:")

datacenter\_destino = st.selectbox("Seleccione el DataCenter de destino:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

rack\_destino = st.text\_input("Rack y ubicación del equipo en el DataCenter de destino:")

# Cantidad de equipos a mover

cantidad

salio el programa cortado, puedes reenviar el codigo?

¡Claro, Roberto! Aquí tienes nuevamente el **código completo y final para** app.py, incluyendo las tres funcionalidades: Checklist de Ingreso, Checklist de Egreso y el nuevo Checklist de Movimiento Interno. Este programa está optimizado con los buscadores de marcas y modelos, validaciones y todas las configuraciones necesarias.

**Código completo**

python

import streamlit as st

import sqlite3

import pandas as pd

# Función para conectar a la base de datos

def conectar\_db():

conn = sqlite3.connect("checklist\_data\_center.db")

return conn

# Función para obtener marcas únicas desde el catálogo

def obtener\_marcas():

conn = conectar\_db()

try:

marcas\_df = pd.read\_sql\_query("SELECT DISTINCT marca FROM catalogo ORDER BY marca ASC", conn)

return marcas\_df["marca"].tolist()

finally:

conn.close()

# Función para obtener modelos en función de la marca seleccionada

def obtener\_modelos(marca\_seleccionada):

conn = conectar\_db()

try:

modelos\_df = pd.read\_sql\_query(

"SELECT DISTINCT modelo FROM catalogo WHERE marca = ? ORDER BY modelo ASC",

conn, params=(marca\_seleccionada,))

return modelos\_df["modelo"].tolist()

finally:

conn.close()

# Función para obtener clientes

def obtener\_clientes():

return ["Cliente A", "Cliente B", "Cliente C"]

# Pantalla inicial para elegir el tipo de checklist

st.title("Checklist de Data Center")

tipo\_checklist = st.selectbox("Seleccione el tipo de checklist:", ["Ingreso", "Egreso", "Movimiento interno"])

# Lógica para el checklist de ingreso

if tipo\_checklist == "Ingreso":

st.subheader("Checklist de ingreso")

# Datos básicos del solicitante

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_ingreso = st.date\_input("Fecha de ingreso:")

cliente = st.selectbox("Seleccione el cliente:", obtener\_clientes())

datacenter = st.selectbox("Seleccione el DataCenter:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

ticket = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):")

# Cantidad de equipos a ingresar

cantidad\_equipos = st.number\_input("Cantidad de equipos a ingresar:", min\_value=1, step=1)

# Campos dinámicos para cada equipo

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

# Buscador de marcas

marca\_busqueda = st.text\_input(f"Buscar marca (Equipo {i}):", key=f"marca\_busqueda\_{i}")

marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()]

marca\_seleccionada = st.selectbox(f"Seleccione la marca del equipo {i}:", marcas, key=f"marca\_{i}")

# Buscador de modelos

modelo\_busqueda = st.text\_input(f"Buscar modelo (Equipo {i}):", key=f"modelo\_busqueda\_{i}")

modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()]

modelo\_seleccionado = st.selectbox(f"Seleccione el modelo del equipo {i}:", modelos, key=f"modelo\_{i}")

# Número de serie

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

equipos.append({"marca": marca\_seleccionada, "modelo": modelo\_seleccionado, "serie": numero\_serie})

# Botón para guardar la solicitud

if st.button("Guardar solicitud de ingreso"):

if all([solicitante, cliente, datacenter, ticket]) and all(e["serie"] for e in equipos):

conn = conectar\_db()

try:

cursor = conn.cursor()

for equipo in equipos:

cursor.execute("""

INSERT INTO solicitudes\_ingreso (

solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

marca, modelo, numero\_serie, tipo\_equipo

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

equipo["marca"], equipo["modelo"], equipo["serie"], "Automático"))

conn.commit()

st.success("¡Solicitud de ingreso guardada correctamente!")

finally:

conn.close()

else:

st.error("Por favor, complete todos los campos obligatorios.")

# Lógica para el checklist de egreso

elif tipo\_checklist == "Egreso":

st.subheader("Checklist de egreso")

# Datos básicos del solicitante

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_egreso = st.date\_input("Fecha de egreso:")

datacenter\_egreso = st.selectbox("Seleccione el DataCenter de egreso:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

ticket\_egreso = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):")

# Cantidad de equipos a egresar

cantidad\_equipos = st.number\_input("Cantidad de equipos a egresar:", min\_value=1, step=1)

# Campos dinámicos para cada equipo

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

# Buscador de marcas

marca\_busqueda = st.text\_input(f"Buscar marca (Equipo {i}):", key=f"marca\_busqueda\_{i}")

marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()]

marca\_seleccionada = st.selectbox(f"Seleccione la marca del equipo {i}:", marcas, key=f"marca\_{i}")

# Buscador de modelos

modelo\_busqueda = st.text\_input(f"Buscar modelo (Equipo {i}):", key=f"modelo\_busqueda\_{i}")

modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()]

modelo\_seleccionado = st.selectbox(f"Seleccione el modelo del equipo {i}:", modelos, key=f"modelo\_{i}")

# Número de serie

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

# Rack de origen

rack\_origen = st.text\_input(f"Rack de origen del equipo {i}:", key=f"rack\_{i}")

# Motivo del egreso

motivo\_egreso = st.selectbox(f"Motivo del egreso (Equipo {i}):", ["Desinstalación", "Traslado", "Otro"], key=f"motivo\_{i}")

# Estado del equipo

estado\_salida = st.selectbox(f"Estado del equipo al salir (Equipo {i}):", ["Funcional", "Dañado"], key=f"estado\_{i}")

equipos.append({

"marca": marca\_seleccionada,

"modelo": modelo\_seleccionado,

"serie": numero\_serie,

"rack\_origen": rack\_origen,

"motivo\_egreso": motivo\_egreso,

"estado\_salida": estado\_salida

})

# Botón para guardar la solicitud

if st.button("Guardar solicitud de egreso"):

if all([solicitante, fecha\_egreso, datacenter\_egreso, ticket\_egreso]) and all(e["serie"] for e in equipos):

conn = conectar\_db()

try:

cursor = conn.cursor()

for equipo in equipos:

cursor.execute("""

INSERT INTO solicitudes\_egreso (

solicitante, fecha\_egreso, datacenter, ticket, marca, modelo, numero\_serie,

rack\_origen, motivo\_egreso, estado\_salida

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_egreso, datacenter\_egreso, ticket\_egreso,

equipo["marca"], equipo["modelo"], equipo["serie"],

equipo["rack\_origen"], equipo["motivo\_egreso"], equipo["estado\_salida"]))

conn.commit()

st.success("¡Solicitud de egreso guardada correctamente!")

finally:

conn.close()

else:

st.error("Por favor, complete todos los campos obligatorios.")

# Lógica para el checklist de movimiento interno

elif tipo\_checklist == "Movimiento interno":

st.subheader("Checklist de movimiento interno")

# Datos básicos del solicitante

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_movimiento = st.date\_input("Fecha del movimiento:")

ticket\_movimiento = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):")

# DataCenters de origen y destino

datacenter\_origen = st.selectbox("Seleccione el DataCenter de origen:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

rack\_origen = st.text\_input("Rack y ubicación del equipo en el DataCenter de origen:")

datacenter\_destino = st.selectbox("Seleccione el DataCenter de destino:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

rack\_destino = st.text\_input("Rack y ubicación del equipo en el Data

que viene despues de esto? # DataCenters de origen y destino datacenter\_origen = st.selectbox("Seleccione el DataCenter de origen:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"]) rack\_origen = st.text\_input("Rack y ubicación del equipo en el DataCenter de origen:") datacenter\_destino = st.selectbox("Seleccione el DataCenter de destino:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"]) rack\_destino = st.text\_input("Rack y ubicación del equipo en el Data

¡Gracias por mencionarlo, Roberto! Aquí está el bloque que continúa después de la sección de **DataCenters de origen y destino** para completar el Checklist de Movimiento Interno:

**Código continuación**

Después del bloque de los DataCenters de origen y destino, se incluye la funcionalidad para gestionar la información de los equipos que serán movidos:

python

# Cantidad de equipos a mover

cantidad\_equipos = st.number\_input("Cantidad de equipos a mover:", min\_value=1, step=1)

# Campos dinámicos para cada equipo

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

# Buscador de marcas

marca\_busqueda = st.text\_input(f"Buscar marca (Equipo {i}):", key=f"marca\_busqueda\_{i}")

marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()]

marca\_seleccionada = st.selectbox(f"Seleccione la marca del equipo {i}:", marcas, key=f"marca\_{i}")

# Buscador de modelos

modelo\_busqueda = st.text\_input(f"Buscar modelo (Equipo {i}):", key=f"modelo\_busqueda\_{i}")

modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()]

modelo\_seleccionado = st.selectbox(f"Seleccione el modelo del equipo {i}:", modelos, key=f"modelo\_{i}")

# Número de serie

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

# Estado del equipo antes del movimiento

estado\_equipo = st.selectbox(f"Estado del equipo antes del movimiento (Equipo {i}):", ["Funcional", "Dañado"], key=f"estado\_equipo\_{i}")

equipos.append({

"marca": marca\_seleccionada,

"modelo": modelo\_seleccionado,

"serie": numero\_serie,

"estado\_equipo": estado\_equipo

})

# Botón para guardar la solicitud

if st.button("Guardar solicitud de movimiento interno"):

if all([solicitante, fecha\_movimiento, ticket\_movimiento, datacenter\_origen, rack\_origen, datacenter\_destino, rack\_destino]) and all(e["serie"] for e in equipos):

conn = conectar\_db()

try:

cursor = conn.cursor()

for equipo in equipos:

cursor.execute("""

INSERT INTO movimientos\_internos (

solicitante, fecha\_movimiento, ticket, datacenter\_origen, rack\_origen,

datacenter\_destino, rack\_destino, marca, modelo, numero\_serie, estado\_equipo

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_movimiento, ticket\_movimiento, datacenter\_origen, rack\_origen,

datacenter\_destino, rack\_destino, equipo["marca"], equipo["modelo"], equipo["serie"], equipo["estado\_equipo"]))

conn.commit()

st.success("¡Solicitud de movimiento interno guardada correctamente!")

finally:

conn.close() # Siempre cierra la conexión

else:

st.error("Por favor, complete todos los campos obligatorios.")

**Descripción de este bloque**

1. **Cantidad de equipos a mover**:
   * Se permite definir el número de equipos que serán transferidos en el movimiento interno.
2. **Campos dinámicos por equipo**:
   * Para cada equipo, se incluye:
     + Buscadores dinámicos de marca y modelo.
     + Número de serie.
     + Estado del equipo antes del movimiento.
3. **Validación de datos obligatorios**:
   * Antes de guardar la solicitud, se verifica que todos los campos estén completos.
4. **Inserción en la base de datos**:
   * Cada equipo y su información se guarda en la tabla movimientos\_internos.

**Pasos adicionales**

Si ya tienes configurada la tabla movimientos\_internos en tu base de datos (como te indiqué previamente en database.py), el código debería funcionar correctamente. Verifica que los datos se registran bien en la base de datos usando **DB Browser for SQLite**.

¿Continuamos con alguna prueba o ajustes adicionales? 😊🚀

me puedes dar el codigo despues de # Lógica para el checklist de movimiento interno

¡Claro, Roberto! Aquí tienes el código completo para la sección que sigue después de # Lógica para el checklist de movimiento interno. Este bloque implementa la funcionalidad completa del formulario de **Movimiento Interno**.

**Código para el checklist de movimiento interno**

python

elif tipo\_checklist == "Movimiento interno":

st.subheader("Checklist de movimiento interno")

# Datos básicos del solicitante

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_movimiento = st.date\_input("Fecha del movimiento:")

ticket\_movimiento = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):")

# DataCenters de origen y destino

datacenter\_origen = st.selectbox("Seleccione el DataCenter de origen:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

rack\_origen = st.text\_input("Rack y ubicación del equipo en el DataCenter de origen:")

datacenter\_destino = st.selectbox("Seleccione el DataCenter de destino:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

rack\_destino = st.text\_input("Rack y ubicación del equipo en el DataCenter de destino:")

# Cantidad de equipos a mover

cantidad\_equipos = st.number\_input("Cantidad de equipos a mover:", min\_value=1, step=1)

# Campos dinámicos para cada equipo

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

# Buscador de marcas

marca\_busqueda = st.text\_input(f"Buscar marca (Equipo {i}):", key=f"marca\_busqueda\_{i}")

marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()]

marca\_seleccionada = st.selectbox(f"Seleccione la marca del equipo {i}:", marcas, key=f"marca\_{i}")

# Buscador de modelos

modelo\_busqueda = st.text\_input(f"Buscar modelo (Equipo {i}):", key=f"modelo\_busqueda\_{i}")

modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()]

modelo\_seleccionado = st.selectbox(f"Seleccione el modelo del equipo {i}:", modelos, key=f"modelo\_{i}")

# Número de serie

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

# Estado del equipo antes del movimiento

estado\_equipo = st.selectbox(f"Estado del equipo antes del movimiento (Equipo {i}):", ["Funcional", "Dañado"], key=f"estado\_equipo\_{i}")

equipos.append({

"marca": marca\_seleccionada,

"modelo": modelo\_seleccionado,

"serie": numero\_serie,

"estado\_equipo": estado\_equipo

})

# Botón para guardar la solicitud

if st.button("Guardar solicitud de movimiento interno"):

if all([solicitante, fecha\_movimiento, ticket\_movimiento, datacenter\_origen, rack\_origen, datacenter\_destino, rack\_destino]) and all(e["serie"] for e in equipos):

conn = conectar\_db()

try:

cursor = conn.cursor()

for equipo in equipos:

cursor.execute("""

INSERT INTO movimientos\_internos (

solicitante, fecha\_movimiento, ticket, datacenter\_origen, rack\_origen,

datacenter\_destino, rack\_destino, marca, modelo, numero\_serie, estado\_equipo

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_movimiento, ticket\_movimiento, datacenter\_origen, rack\_origen,

datacenter\_destino, rack\_destino, equipo["marca"], equipo["modelo"], equipo["serie"], equipo["estado\_equipo"]))

conn.commit()

st.success("¡Solicitud de movimiento interno guardada correctamente!")

finally:

conn.close() # Siempre cierra la conexión

else:

st.error("Por favor, complete todos los campos obligatorios.")

**Descripción del código**

1. **Formulario dinámico:**
   * **Solicitante:** Recolecta datos del solicitante, como el nombre y el ticket asociado.
   * **DataCenter y ubicación:** Captura la información de origen y destino del movimiento del equipo.
   * **Equipos:** Permite especificar múltiples equipos, con detalles como marca, modelo, número de serie, y estado.
2. **Validación de datos:**
   * Antes de guardar, verifica que todos los campos estén completos y que se hayan ingresado todos los equipos.
3. **Inserción en la base de datos:**
   * Registra cada equipo individualmente en la tabla movimientos\_internos.

**Requisitos adicionales**

Asegúrate de que la tabla movimientos\_internos esté creada en tu base de datos. Para hacerlo, usa este código en tu archivo database.py:

python

cursor.execute("DROP TABLE IF EXISTS movimientos\_internos")

cursor.execute("""

CREATE TABLE IF NOT EXISTS movimientos\_internos (

id INTEGER PRIMARY KEY AUTOINCREMENT,

solicitante TEXT NOT NULL,

fecha\_movimiento TEXT NOT NULL,

ticket TEXT NOT NULL,

datacenter\_origen TEXT NOT NULL,

rack\_origen TEXT,

datacenter\_destino TEXT NOT NULL,

rack\_destino TEXT,

marca TEXT NOT NULL,

modelo TEXT NOT NULL,

numero\_serie TEXT NOT NULL,

estado\_equipo TEXT NOT NULL

)

""")

Ejecuta este código en database.py para garantizar que la tabla esté lista antes de probar el formulario.

**Pruebas sugeridas**

1. **Completa el formulario:**
   * Ingresa información del solicitante, DataCenters, y detalles de equipos.
2. **Guarda la solicitud:**
   * Asegúrate de que todos los datos se guardan correctamente en la base de datos.
3. **Verifica con DB Browser for SQLite:**
   * Abre la tabla movimientos\_internos y confirma que los datos aparecen correctamente.

¡Dime si hay algo más en lo que pueda ayudarte o si quieres continuar con otro paso del proyecto! 🚀😊

quedo perfecto, sigamos

¡Genial, Roberto! Me alegra que todo haya quedado como querías. 🚀 Vamos avanzando juntos.

Dime, ¿qué te gustaría hacer ahora? Aquí tienes algunas opciones para continuar:

1. **Generación de reportes**: Implementar exportación de datos en formatos como Excel o PDF para reportes del Checklist.
2. **Visualización de datos**: Crear gráficos o tablas dinámicas para analizar las solicitudes registradas.
3. **Gestión avanzada**: Agregar funciones para buscar, editar o eliminar registros desde la interfaz.
4. **Optimización o nuevas funcionalidades**: Si tienes algo más en mente, lo podemos diseñar.

¿Qué te parece? 😊

antes de continuar, todo lo que hemos avanzado, lo puedes agregar a un archivo word?

Por ahora no tengo la capacidad de crear archivos Word directamente desde esta interfaz. Sin embargo, puedo ayudarte a organizar todo lo que hemos avanzado para que puedas copiar y pegar el contenido en Word de manera rápida y sencilla.

Aquí te dejo una estructura organizada con todo lo que hemos trabajado:

**1. Introducción**

Resumen del proyecto:

* Implementación de checklists dinámicos para Ingreso, Egreso y Movimiento Interno de equipos en el Data Center.
* Funcionalidades avanzadas como buscadores de marcas y modelos, validaciones y conexión a base de datos SQLite.

**2. Código completo**

**Checklist de Ingreso**

python

# Lógica del checklist de ingreso

<Coloca aquí el código completo del Checklist de Ingreso>

**Checklist de Egreso**

python

# Lógica del checklist de egreso

<Coloca aquí el código completo del Checklist de Egreso>

**Checklist de Movimiento Interno**

python

# Lógica del checklist de movimiento interno

<Coloca aquí el código completo del Checklist de Movimiento Interno>

**3. Base de datos**

**Estructura de tablas**

sql

# Tabla solicitudes\_ingreso

CREATE TABLE IF NOT EXISTS solicitudes\_ingreso (

id INTEGER PRIMARY KEY AUTOINCREMENT,

solicitante TEXT NOT NULL,

fecha\_ingreso TEXT NOT NULL,

cliente TEXT NOT NULL,

datacenter TEXT NOT NULL,

ticket TEXT NOT NULL,

cantidad\_equipos INTEGER NOT NULL,

marca TEXT NOT NULL,

modelo TEXT NOT NULL,

numero\_serie TEXT NOT NULL,

tipo\_equipo TEXT NOT NULL

);

# Tabla solicitudes\_egreso

CREATE TABLE IF NOT EXISTS solicitudes\_egreso (

id INTEGER PRIMARY KEY AUTOINCREMENT,

solicitante TEXT NOT NULL,

fecha\_egreso TEXT NOT NULL,

datacenter TEXT NOT NULL,

ticket TEXT NOT NULL,

marca TEXT NOT NULL,

modelo TEXT NOT NULL,

numero\_serie TEXT NOT NULL,

rack\_origen TEXT,

motivo\_egreso TEXT NOT NULL,

estado\_salida TEXT NOT NULL

);

# Tabla movimientos\_internos

CREATE TABLE IF NOT EXISTS movimientos\_internos (

id INTEGER PRIMARY KEY AUTOINCREMENT,

solicitante TEXT NOT NULL,

fecha\_movimiento TEXT NOT NULL,

ticket TEXT NOT NULL,

datacenter\_origen TEXT NOT NULL,

rack\_origen TEXT,

datacenter\_destino TEXT NOT NULL,

rack\_destino TEXT,

marca TEXT NOT NULL,

modelo TEXT NOT NULL,

numero\_serie TEXT NOT NULL,

estado\_equipo TEXT NOT NULL

);

Si copias esta información y la pegas en un archivo Word, quedará perfectamente organizado. ¿Te gustaría que te ayude con algún formato específico, como incluir tablas o listas detalladas? 😊

ok seguiremos en un rato mas, gracias

¡Por supuesto, Roberto! Estoy aquí cuando quieras continuar. Que tengas un buen rato mientras tanto 😊🚀.

continuemos con el proyecto

¡Excelente, Roberto! Vamos a seguir avanzando juntos. 🚀 ¿Qué te gustaría agregar o mejorar en el proyecto ahora? Aquí hay algunas ideas para inspirarte:

1. **Generación de reportes:** Automatizar la creación de reportes en formatos como Excel o PDF, basados en los datos registrados en los checklists.
2. **Visualización avanzada:** Crear gráficos o tablas interactivas en Streamlit para analizar las solicitudes registradas.
3. **Gestión avanzada:** Agregar funciones para buscar, editar o eliminar registros desde la interfaz.
4. **Validaciones adicionales:** Fortalecer las reglas de entrada de datos para evitar errores.
5. **Notificaciones o correos electrónicos:** Integrar envíos automáticos de confirmación cuando se registra un movimiento interno, ingreso o egreso.

Dime cuál te interesa más o si tienes algo distinto en mente. ¡Vamos con todo! 😊

empecemos con todo lo que me indicas. esto es antes de lo que habiamos indicado en la respuesta y las alertas por correo de nueva solicitud ingresada?

¡Perfecto, Roberto! Vamos a trabajar en todas estas funcionalidades desde ahora. Aquí te detallo cómo vamos a organizarnos para abordar cada punto:

**1. Generación de reportes**

Antes que nada, implementaremos la exportación de datos a un archivo Excel para que puedas generar reportes basados en las solicitudes registradas (Ingreso, Egreso y Movimiento Interno).

**2. Visualización avanzada**

Crearemos gráficos y tablas interactivas directamente en Streamlit para analizar los datos registrados, como:

* Número de equipos ingresados por DataCenter.
* Motivos de egreso más frecuentes.
* Movimientos internos por DataCenter origen/destino.

**3. Gestión avanzada**

Agregaremos funcionalidades para buscar, editar o eliminar registros:

* Búsqueda por nombre de solicitante, ticket o fecha.
* Edición y eliminación de registros específicos desde la interfaz.

**4. Validaciones adicionales**

Mejoraremos la entrada de datos para prevenir errores:

* Verificación en tiempo real (por ejemplo, asegurarnos de que el número de serie no esté vacío).
* Alertas cuando se registren equipos duplicados en una solicitud.

**5. Notificaciones o alertas por correo**

Configuraremos envíos automáticos de correos electrónicos para notificar sobre nuevas solicitudes ingresadas. Esto incluye:

* Configurar una cuenta de correo electrónico (por ejemplo, SMTP con Gmail o Outlook).
* Enviar un correo cada vez que se registre una solicitud con un resumen de los datos.

**Orden sugerido**

Vamos en el siguiente orden:

1. **Generación de reportes (Excel).**
2. **Visualización avanzada (gráficos y tablas dinámicas).**
3. **Gestión avanzada (buscar, editar, eliminar).**
4. **Validaciones adicionales.**
5. **Alertas por correo.**

Dime si te parece bien este enfoque o si quieres comenzar directamente por algún punto en específico. ¡Listos para avanzar! 😊🚀

si para continuar con esto vamos paso a paso con los puntos indicados, ya te dire despues en que continuaremos. recuerda decirme los pasos detalle a detalle

Perfecto, Roberto. Vamos paso a paso y explicaré cada punto con detalle para que implementes todo fácilmente. Comencemos con el **primer punto: Generación de reportes**. Te daré una guía detallada para implementar la exportación de datos en Excel utilizando la librería **pandas**.

**1. Generación de reportes (Excel)**

**Objetivo:**

Crear la funcionalidad para exportar los datos registrados en las tablas solicitudes\_ingreso, solicitudes\_egreso, y movimientos\_internos a archivos Excel desde Streamlit.

**Paso 1: Instalar las librerías necesarias**

1. Asegúrate de tener la librería pandas instalada (que ya usamos en el proyecto) y también instala **openpyxl**, que permite generar archivos Excel:

bash

pip install openpyxl

**Paso 2: Función para exportar datos a Excel**

Agrega esta función en tu archivo app.py, antes de los checklists:

python

def exportar\_a\_excel(tabla, nombre\_archivo):

conn = conectar\_db()

try:

# Leer los datos de la tabla especificada

query = f"SELECT \* FROM {tabla}"

df = pd.read\_sql\_query(query, conn)

# Exportar los datos a un archivo Excel

df.to\_excel(nombre\_archivo, index=False, engine='openpyxl')

return f"Archivo {nombre\_archivo} generado correctamente."

finally:

conn.close()

**Paso 3: Agregar un botón para exportar los datos**

Dentro de tu aplicación de Streamlit, agrega la opción para exportar los datos de cada tabla. Por ejemplo, puedes colocar esta lógica después de los checklists:

python

if st.checkbox("Generar reporte"):

tabla\_seleccionada = st.selectbox("Seleccione la tabla para exportar:", ["solicitudes\_ingreso", "solicitudes\_egreso", "movimientos\_internos"])

nombre\_archivo = st.text\_input("Nombre del archivo Excel (ejemplo: reporte.xlsx):")

if st.button("Exportar"):

if nombre\_archivo:

resultado = exportar\_a\_excel(tabla\_seleccionada, nombre\_archivo)

st.success(resultado)

else:

st.error("Por favor, ingrese un nombre de archivo.")

**Paso 4: Probar la funcionalidad**

1. Ejecuta la aplicación:

bash

streamlit run app.py

1. Marca la casilla **Generar reporte** en la interfaz.
2. Selecciona una tabla (por ejemplo, solicitudes\_ingreso) y proporciona un nombre de archivo Excel (por ejemplo, reporte\_ingreso.xlsx).
3. Presiona el botón **Exportar** y verifica que el archivo se haya generado correctamente en tu carpeta de proyecto.

**Siguientes pasos**

Una vez que completes la generación de reportes, podemos avanzar al siguiente punto: **Visualización avanzada con gráficos y tablas dinámicas en Streamlit**.

¿Te gustaría que explique cómo hacerlo o prefieres terminar primero esta parte? 😊🚀

continuemos

¡Excelente, Roberto! Ahora que ya abordamos la generación de reportes, continuemos con **Visualización avanzada**. Aquí trabajaremos para agregar gráficos y tablas dinámicas directamente en Streamlit para analizar los datos registrados.

**2. Visualización avanzada**

**Objetivo:**

Mostrar estadísticas y resúmenes visuales de las tablas en la base de datos con gráficos y tablas interactivas. Ejemplo:

* Número de equipos ingresados por DataCenter.
* Distribución de motivos de egreso.
* Movimientos internos por DataCenter origen/destino.

**Paso 1: Crear una sección de visualización**

Agregaremos una nueva sección en la aplicación para análisis interactivo. Coloca este código en app.py después de los checklists:

python

if st.checkbox("Visualizar datos"):

tabla\_seleccionada = st.selectbox("Seleccione la tabla para analizar:", ["solicitudes\_ingreso", "solicitudes\_egreso", "movimientos\_internos"])

# Conexión a la base de datos

conn = conectar\_db()

try:

# Cargar los datos de la tabla seleccionada

query = f"SELECT \* FROM {tabla\_seleccionada}"

df = pd.read\_sql\_query(query, conn)

# Mostrar los datos en una tabla interactiva

st.subheader("Tabla de datos")

st.dataframe(df)

# Generar gráficos según la tabla seleccionada

if tabla\_seleccionada == "solicitudes\_ingreso":

st.subheader("Análisis de Ingresos")

grafico\_ingresos = df["datacenter"].value\_counts()

st.bar\_chart(grafico\_ingresos)

elif tabla\_seleccionada == "solicitudes\_egreso":

st.subheader("Análisis de Egresos")

grafico\_egresos = df["motivo\_egreso"].value\_counts()

st.bar\_chart(grafico\_egresos)

elif tabla\_seleccionada == "movimientos\_internos":

st.subheader("Análisis de Movimientos Internos")

grafico\_movimientos = df["datacenter\_origen"].value\_counts()

st.bar\_chart(grafico\_movimientos)

finally:

conn.close()

**Paso 2: Explicación del código**

1. **Selección de tabla:**
   * El usuario selecciona la tabla que desea analizar (Ingreso, Egreso o Movimiento Interno).
2. **Tabla interactiva:**
   * Los datos de la tabla seleccionada se muestran en un formato interactivo.
3. **Gráficos dinámicos:**
   * Según la tabla seleccionada, se generan gráficos relevantes:
     + **Ingresos:** Cantidad de equipos ingresados por DataCenter.
     + **Egresos:** Distribución de motivos de egreso.
     + **Movimientos Internos:** Movimientos internos agrupados por DataCenter de origen.

**Paso 3: Probar la funcionalidad**

1. Ejecuta la aplicación:

bash

streamlit run app.py

1. Marca la casilla **Visualizar datos**.
2. Selecciona una tabla para análisis interactivo.
3. Verifica que la tabla y los gráficos se muestran correctamente en la interfaz.

**Siguientes pasos**

Cuando terminemos la visualización avanzada, podremos continuar con la **Gestión avanzada** para buscar, editar o eliminar registros. Dime si hay algo adicional que quieres ajustar en este paso o seguimos con el siguiente. ¡Avanzamos juntos! 😊🚀

continuemos

¡Vamos a por el siguiente paso, Roberto! Ahora trabajaremos en la **Gestión avanzada** para buscar, editar y eliminar registros desde la interfaz. Esto permitirá que tengas un control completo sobre los datos en tu base de datos. 🚀

**3. Gestión avanzada**

**Objetivo:**

Agregar funcionalidades para:

1. **Buscar registros**: Permite encontrar datos específicos según diferentes criterios (como solicitante, ticket, fecha, etc.).
2. **Editar registros**: Actualiza la información directamente desde la interfaz.
3. **Eliminar registros**: Elimina un registro específico de la base de datos.

**Paso 1: Crear la sección de gestión avanzada**

En tu archivo app.py, agrega una nueva sección para la gestión de registros. Coloca este bloque después de las visualizaciones de datos:

python

if st.checkbox("Gestión de registros"):

tabla\_seleccionada = st.selectbox("Seleccione la tabla para gestionar:", ["solicitudes\_ingreso", "solicitudes\_egreso", "movimientos\_internos"])

conn = conectar\_db()

try:

# Cargar los datos de la tabla seleccionada

query = f"SELECT \* FROM {tabla\_seleccionada}"

df = pd.read\_sql\_query(query, conn)

st.subheader("Registros disponibles")

st.dataframe(df)

# Buscar registros

st.subheader("Buscar registros")

campo\_busqueda = st.selectbox("Seleccione el campo para buscar:", df.columns)

valor\_busqueda = st.text\_input("Ingrese el valor para buscar:")

if st.button("Buscar"):

resultados = df[df[campo\_busqueda].astype(str).str.contains(valor\_busqueda, case=False, na=False)]

if not resultados.empty:

st.write("Resultados encontrados:")

st.dataframe(resultados)

else:

st.warning("No se encontraron registros con ese criterio.")

# Editar registros

st.subheader("Editar registros")

id\_registro = st.number\_input("Ingrese el ID del registro que desea editar:", min\_value=1, step=1)

columna\_editar = st.selectbox("Seleccione la columna que desea editar:", df.columns)

nuevo\_valor = st.text\_input("Ingrese el nuevo valor:")

if st.button("Guardar cambios"):

cursor = conn.cursor()

cursor.execute(f"UPDATE {tabla\_seleccionada} SET {columna\_editar} = ? WHERE id = ?", (nuevo\_valor, id\_registro))

conn.commit()

st.success("Registro actualizado correctamente.")

# Eliminar registros

st.subheader("Eliminar registros")

id\_eliminar = st.number\_input("Ingrese el ID del registro que desea eliminar:", min\_value=1, step=1)

if st.button("Eliminar"):

cursor = conn.cursor()

cursor.execute(f"DELETE FROM {tabla\_seleccionada} WHERE id = ?", (id\_eliminar,))

conn.commit()

st.success("Registro eliminado correctamente.")

finally:

conn.close()

**Paso 2: Explicación del código**

1. **Cargar registros:**
   * Los datos de la tabla seleccionada se muestran en un formato interactivo.
2. **Buscar registros:**
   * El usuario puede buscar por cualquier columna seleccionada y ver resultados en tiempo real.
3. **Editar registros:**
   * Permite actualizar valores específicos de un registro según el ID proporcionado.
4. **Eliminar registros:**
   * Permite eliminar cualquier registro seleccionando su ID.

**Paso 3: Probar la funcionalidad**

1. Ejecuta la aplicación:

bash

streamlit run app.py

1. Marca la casilla **Gestión de registros**.
2. Prueba cada funcionalidad:
   * Buscar por cualquier columna y criterio.
   * Editar un registro existente y verificar los cambios en la base de datos.
   * Eliminar un registro y confirmarlo en la base de datos.

**Siguientes pasos**

¿Te gustaría ajustar algo en la gestión avanzada o avanzamos con el siguiente punto: **Validaciones adicionales**? ¡Estamos haciendo grandes progresos! 😊🚀

continuemos

¡Muy bien, Roberto! El siguiente paso es trabajar en las **Validaciones adicionales** para fortalecer la entrada de datos, evitar errores comunes y garantizar que todo lo que se registre sea correcto y consistente.

**4. Validaciones adicionales**

**Objetivo:**

1. Prevenir errores de entrada al verificar los datos ingresados en tiempo real.
2. Asegurarnos de que no se ingresen datos duplicados o incompletos (por ejemplo, el número de serie de un equipo ya registrado).
3. Ofrecer mensajes de alerta claros cuando los datos no sean válidos.

**Paso 1: Validar datos en los checklists**

Dentro de cada uno de los checklists (Ingreso, Egreso, Movimiento Interno), agregaremos validaciones específicas para asegurar que los datos cumplan ciertos criterios. Aquí tienes ejemplos de validaciones que podemos agregar:

**Validación de duplicados (número de serie)**

Asegúrate de que no se ingrese el mismo número de serie más de una vez:

python

# Verificar que los números de serie no sean duplicados

numeros\_serie = [equipo["serie"] for equipo in equipos]

if len(numeros\_serie) != len(set(numeros\_serie)):

st.error("Hay números de serie duplicados en los equipos. Verifique e intente nuevamente.")

**Validación de campos obligatorios**

Antes de guardar los datos, valida que todos los campos estén llenos:

python

if not solicitante or not fecha\_ingreso or not cliente or not datacenter or not ticket:

st.error("Todos los campos obligatorios deben ser completados.")

**Paso 2: Validación contra la base de datos**

Evitar registros duplicados en la base de datos, como un número de serie que ya existe. Agrega esta validación antes de insertar datos:

python

def verificar\_duplicado(tabla, campo, valor):

conn = conectar\_db()

try:

query = f"SELECT COUNT(\*) FROM {tabla} WHERE {campo} = ?"

cursor = conn.cursor()

cursor.execute(query, (valor,))

resultado = cursor.fetchone()[0]

return resultado > 0

finally:

conn.close()

Usa esta función antes de guardar un registro:

python

if verificar\_duplicado("solicitudes\_ingreso", "numero\_serie", equipo["serie"]):

st.error(f"El número de serie {equipo['serie']} ya está registrado en la base de datos.")

**Paso 3: Implementar validaciones específicas por checklist**

Aquí hay ejemplos de dónde aplicar estas validaciones:

**Checklist de Ingreso**

* Valida que cada equipo tenga un número de serie único.
* Verifica que el ticket no esté vacío y, si es "Pendiente", solicita confirmación del usuario.

**Checklist de Egreso**

* Confirma que el rack de origen no esté vacío.
* Valida que el número de serie exista en la base de datos antes de egresarlo.

**Checklist de Movimiento Interno**

* Verifica que el DataCenter de origen y destino no sean iguales.
* Confirma que los equipos seleccionados estén registrados en la tabla de ingreso.

**Paso 4: Mostrar mensajes claros al usuario**

Asegúrate de que los errores o advertencias sean entendibles y amigables. Por ejemplo:

python

st.error("Por favor, asegúrese de que no hay equipos duplicados en la lista.")

st.warning("El ticket está marcado como 'Pendiente'. ¿Está seguro de continuar?")

st.success("¡Solicitud registrada correctamente!")

**Paso 5: Probar las validaciones**

1. Intenta registrar datos incorrectos (como números de serie duplicados) y verifica que aparezcan los mensajes de error correspondientes.
2. Registra un número de serie ya existente en la base de datos y confirma que no se permita la operación.
3. Prueba con datos válidos y asegúrate de que se guarden correctamente.

**Siguientes pasos**

Cuando terminemos con estas validaciones, podemos avanzar al último punto: **Alertas por correo**. ¿Quieres implementar algo más en las validaciones antes de continuar? 😊🚀

indicame donde coloco los codigos anteriores, para entender un poco

¡Claro, Roberto! Te explicaré en detalle dónde colocar los códigos anteriores y cómo integrarlos en tu proyecto de Streamlit para que todo tenga sentido. 😊

**Organización de los códigos**

**1. Función para verificar duplicados**

Este código debe estar ubicado en la sección de **funciones auxiliares**, generalmente al inicio de tu archivo app.py, junto con otras funciones como conectar\_db().

python

def verificar\_duplicado(tabla, campo, valor):

conn = conectar\_db()

try:

query = f"SELECT COUNT(\*) FROM {tabla} WHERE {campo} = ?"

cursor = conn.cursor()

cursor.execute(query, (valor,))

resultado = cursor.fetchone()[0]

return resultado > 0

finally:

conn.close()

Esta función puede ser llamada en los checklists antes de guardar datos para asegurarse de que no haya duplicados. Por ejemplo, si estás validando un número de serie en el **Checklist de Ingreso**, debes usarla justo antes del comando INSERT INTO.

**2. Validación de números de serie duplicados (en el formulario dinámico)**

Esta validación debe ser colocada dentro de las secciones correspondientes a los **checklists dinámicos**, como el de **Ingreso**, **Egreso** o **Movimiento Interno**, donde estás definiendo los equipos que se registran. Por ejemplo, en el **Checklist de Ingreso**, puedes añadir esta validación después de recoger la lista de equipos:

python

# Verificar que los números de serie no sean duplicados

numeros\_serie = [equipo["serie"] for equipo in equipos]

if len(numeros\_serie) != len(set(numeros\_serie)):

st.error("Hay números de serie duplicados en los equipos. Verifique e intente nuevamente.")

Ubícala justo antes de procesar los datos en el comando INSERT INTO.

**3. Validación de campos obligatorios**

Este código también debe estar dentro de las secciones correspondientes a cada checklist. Colócalo antes de permitir que los datos sean enviados a la base de datos. Por ejemplo:

python

if not solicitante or not fecha\_ingreso or not cliente or not datacenter or not ticket:

st.error("Todos los campos obligatorios deben ser completados.")

Ubicación sugerida:

* En el **Checklist de Ingreso**, justo antes del botón **Guardar solicitud de ingreso**.
* En el **Checklist de Egreso**, antes del botón **Guardar solicitud de egreso**.
* En el **Checklist de Movimiento Interno**, antes del botón **Guardar solicitud de movimiento interno**.

**4. Validación contra la base de datos (ejemplo de uso)**

Usa la función verificar\_duplicado() dentro de cada checklist para evitar registros duplicados en la base de datos. Por ejemplo, si estás validando el número de serie en el **Checklist de Movimiento Interno**, puedes colocarla antes de insertar los equipos:

python

if verificar\_duplicado("movimientos\_internos", "numero\_serie", equipo["serie"]):

st.error(f"El número de serie {equipo['serie']} ya está registrado en la base de datos.")

Ubicación sugerida:

* En el bloque donde procesas los datos y realizas las inserciones (por ejemplo, antes del INSERT INTO).

**Un ejemplo completo con validaciones integradas**

Te muestro cómo quedaría el **Checklist de Movimiento Interno** con todas las validaciones aplicadas:

python

elif tipo\_checklist == "Movimiento interno":

st.subheader("Checklist de movimiento interno")

# Datos básicos del solicitante

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_movimiento = st.date\_input("Fecha del movimiento:")

ticket\_movimiento = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):")

# DataCenters de origen y destino

datacenter\_origen = st.selectbox("Seleccione el DataCenter de origen:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

rack\_origen = st.text\_input("Rack y ubicación del equipo en el DataCenter de origen:")

datacenter\_destino = st.selectbox("Seleccione el DataCenter de destino:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

rack\_destino = st.text\_input("Rack y ubicación del equipo en el DataCenter de destino:")

# Cantidad de equipos a mover

cantidad\_equipos = st.number\_input("Cantidad de equipos a mover:", min\_value=1, step=1)

# Campos dinámicos para cada equipo

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

# Buscador de marcas

marca\_busqueda = st.text\_input(f"Buscar marca (Equipo {i}):", key=f"marca\_busqueda\_{i}")

marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()]

marca\_seleccionada = st.selectbox(f"Seleccione la marca del equipo {i}:", marcas, key=f"marca\_{i}")

# Buscador de modelos

modelo\_busqueda = st.text\_input(f"Buscar modelo (Equipo {i}):", key=f"modelo\_busqueda\_{i}")

modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()]

modelo\_seleccionado = st.selectbox(f"Seleccione el modelo del equipo {i}:", modelos, key=f"modelo\_{i}")

# Número de serie

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

# Estado del equipo antes del movimiento

estado\_equipo = st.selectbox(f"Estado del equipo antes del movimiento (Equipo {i}):", ["Funcional", "Dañado"], key=f"estado\_equipo\_{i}")

equipos.append({

"marca": marca\_seleccionada,

"modelo": modelo\_seleccionado,

"serie": numero\_serie,

"estado\_equipo": estado\_equipo

})

# Validación de duplicados

numeros\_serie = [equipo["serie"] for equipo in equipos]

if len(numeros\_serie) != len(set(numeros\_serie)):

st.error("Hay números de serie duplicados en los equipos. Verifique e intente nuevamente.")

# Validación de campos obligatorios

if not solicitante or not fecha\_movimiento or not ticket\_movimiento or not datacenter\_origen or not rack\_origen or not datacenter\_destino or not rack\_destino:

st.error("Todos los campos obligatorios deben ser completados.")

# Botón para guardar la solicitud

if st.button("Guardar solicitud de movimiento interno"):

conn = conectar\_db()

try:

cursor = conn.cursor()

for equipo in equipos:

if verificar\_duplicado("movimientos\_internos", "numero\_serie", equipo["serie"]):

st.error(f"El número de serie {equipo['serie']} ya está registrado en la base de datos.")

else:

cursor.execute("""

INSERT INTO movimientos\_internos (

solicitante, fecha\_movimiento, ticket, datacenter\_origen, rack\_origen,

datacenter\_destino, rack\_destino, marca, modelo, numero\_serie, estado\_equipo

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_movimiento, ticket\_movimiento, datacenter\_origen, rack\_origen,

datacenter\_destino, rack\_destino, equipo["marca"], equipo["modelo"], equipo["serie"], equipo["estado\_equipo"]))

conn.commit()

st.success("¡Solicitud de movimiento interno guardada correctamente!")

finally:

conn.close()

Dime si esto te queda claro o si necesitas más detalles en algún punto específico. ¡Vamos avanzando juntos, Roberto! 😊🚀

Hoy

muestrame el ejemplo completo de checklist ingreso y de egreso

¡Por supuesto, Roberto! Aquí tienes el ejemplo completo y actualizado de los **checklists de Ingreso y Egreso**, integrando las validaciones y funcionalidades necesarias.

**Checklist de Ingreso**

python

elif tipo\_checklist == "Ingreso":

st.subheader("Checklist de ingreso")

# Datos básicos del solicitante

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_ingreso = st.date\_input("Fecha de ingreso:")

cliente = st.selectbox("Seleccione el cliente:", obtener\_clientes())

datacenter = st.selectbox("Seleccione el DataCenter:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

ticket = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):")

# Cantidad de equipos a ingresar

cantidad\_equipos = st.number\_input("Cantidad de equipos a ingresar:", min\_value=1, step=1)

# Campos dinámicos para cada equipo

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

# Buscador de marcas

marca\_busqueda = st.text\_input(f"Buscar marca (Equipo {i}):", key=f"marca\_busqueda\_{i}")

marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()]

marca\_seleccionada = st.selectbox(f"Seleccione la marca del equipo {i}:", marcas, key=f"marca\_{i}")

# Buscador de modelos

modelo\_busqueda = st.text\_input(f"Buscar modelo (Equipo {i}):", key=f"modelo\_busqueda\_{i}")

modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()]

modelo\_seleccionado = st.selectbox(f"Seleccione el modelo del equipo {i}:", modelos, key=f"modelo\_{i}")

# Número de serie

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

equipos.append({"marca": marca\_seleccionada, "modelo": modelo\_seleccionado, "serie": numero\_serie})

# Validación de duplicados

numeros\_serie = [equipo["serie"] for equipo in equipos]

if len(numeros\_serie) != len(set(numeros\_serie)):

st.error("Hay números de serie duplicados en los equipos. Verifique e intente nuevamente.")

# Validación de campos obligatorios

if not solicitante or not fecha\_ingreso or not cliente or not datacenter or not ticket:

st.error("Todos los campos obligatorios deben ser completados.")

# Botón para guardar la solicitud

if st.button("Guardar solicitud de ingreso"):

if all([solicitante, cliente, datacenter, ticket]) and all(e["serie"] for e in equipos):

conn = conectar\_db()

try:

cursor = conn.cursor()

for equipo in equipos:

if verificar\_duplicado("solicitudes\_ingreso", "numero\_serie", equipo["serie"]):

st.error(f"El número de serie {equipo['serie']} ya está registrado en la base de datos.")

else:

cursor.execute("""

INSERT INTO solicitudes\_ingreso (

solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

marca, modelo, numero\_serie, tipo\_equipo

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

equipo["marca"], equipo["modelo"], equipo["serie"], "Automático"))

conn.commit()

st.success("¡Solicitud de ingreso guardada correctamente!")

finally:

conn.close()

**Checklist de Egreso**

python

elif tipo\_checklist == "Egreso":

st.subheader("Checklist de egreso")

# Datos básicos del solicitante

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_egreso = st.date\_input("Fecha de egreso:")

datacenter\_egreso = st.selectbox("Seleccione el DataCenter de egreso:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

ticket\_egreso = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):")

# Cantidad de equipos a egresar

cantidad\_equipos = st.number\_input("Cantidad de equipos a egresar:", min\_value=1, step=1)

# Campos dinámicos para cada equipo

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

# Buscador de marcas

marca\_busqueda = st.text\_input(f"Buscar marca (Equipo {i}):", key=f"marca\_busqueda\_{i}")

marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()]

marca\_seleccionada = st.selectbox(f"Seleccione la marca del equipo {i}:", marcas, key=f"marca\_{i}")

# Buscador de modelos

modelo\_busqueda = st.text\_input(f"Buscar modelo (Equipo {i}):", key=f"modelo\_busqueda\_{i}")

modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()]

modelo\_seleccionado = st.selectbox(f"Seleccione el modelo del equipo {i}:", modelos, key=f"modelo\_{i}")

# Número de serie

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

# Rack de origen

rack\_origen = st.text\_input(f"Rack de origen del equipo {i}:", key=f"rack\_{i}")

# Motivo del egreso

motivo\_egreso = st.selectbox(f"Motivo del egreso (Equipo {i}):", ["Desinstalación", "Traslado", "Otro"], key=f"motivo\_{i}")

# Estado del equipo

estado\_salida = st.selectbox(f"Estado del equipo al salir (Equipo {i}):", ["Funcional", "Dañado"], key=f"estado\_{i}")

equipos.append({

"marca": marca\_seleccionada,

"modelo": modelo\_seleccionado,

"serie": numero\_serie,

"rack\_origen": rack\_origen,

"motivo\_egreso": motivo\_egreso,

"estado\_salida": estado\_salida

})

# Validación de campos obligatorios

if not solicitante or not fecha\_egreso or not datacenter\_egreso or not ticket\_egreso:

st.error("Todos los campos obligatorios deben ser completados.")

# Botón para guardar la solicitud

if st.button("Guardar solicitud de egreso"):

if all([solicitante, fecha\_egreso, datacenter\_egreso, ticket\_egreso]) and all(e["serie"] for e in equipos):

conn = conectar\_db()

try:

cursor = conn.cursor()

for equipo in equipos:

cursor.execute("""

INSERT INTO solicitudes\_egreso (

solicitante, fecha\_egreso, datacenter, ticket, marca, modelo, numero\_serie,

rack\_origen, motivo\_egreso, estado\_salida

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_egreso, datacenter\_egreso, ticket\_egreso,

equipo["marca"], equipo["modelo"], equipo["serie"],

equipo["rack\_origen"], equipo["motivo\_egreso"], equipo["estado\_salida"]))

conn.commit()

st.success("¡Solicitud de egreso guardada correctamente!")

finally:

conn.close()

**Resumen de funcionalidades incluidas**

1. **Campos dinámicos**: Se generan formularios para múltiples equipos por checklist.
2. **Validaciones**:
   * Duplicados de número de serie.
   * Campos obligatorios.
3. **Inserción en base de datos**: Los registros se guardan adecuadamente.
4. **Mensajes claros**: Alertas de error y éxito para el usuario.

¡Avísame si quieres ajustar o agregar algo más antes de seguir con el proyecto! 🚀😊

continuemos, por ahora vamos impecable

¡Excelente, Roberto! Me alegra muchísimo que todo esté saliendo perfecto. Ahora seguimos con el último paso que habíamos planificado: **Alertas por correo electrónico**. Este es un gran agregado para automatizar notificaciones cuando se registra una nueva solicitud.

**5. Alertas por correo electrónico**

**Objetivo:**

Enviar un correo electrónico automáticamente cuando se registre una nueva solicitud (ya sea de Ingreso, Egreso o Movimiento Interno) con un resumen de la información ingresada.

**Paso 1: Configurar el envío de correos**

Usaremos la librería integrada de Python llamada smtplib para enviar correos. Si aún no tienes una cuenta de correo configurada, puedes usar Gmail o Outlook. Primero, necesitamos habilitar el acceso SMTP.

**Para Gmail:**

1. Habilita el acceso de aplicaciones menos seguras:
   * Entra a tu cuenta de Google y habilita el acceso desde aquí.
   * Nota: Si tienes la verificación en dos pasos activa, deberás generar una contraseña de aplicación.

**Para Outlook:**

El servidor SMTP de Outlook funciona con tus credenciales normales, y no requiere configuraciones adicionales.

**Paso 2: Crear la función para enviar correos**

Agrega esta función a tu archivo app.py, en la sección de funciones auxiliares:

python

import smtplib

from email.mime.text import MIMEText

from email.mime.multipart import MIMEMultipart

def enviar\_correo(destinatario, asunto, mensaje):

# Configuración del servidor de correo

servidor = "smtp.gmail.com" # Cambia a "smtp.office365.com" si usas Outlook

puerto = 587

remitente = "tu\_correo@gmail.com" # Reemplaza con tu correo

contraseña = "tu\_contraseña" # Reemplaza con tu contraseña o contraseña de aplicación

# Crear el mensaje

email = MIMEMultipart()

email["From"] = remitente

email["To"] = destinatario

email["Subject"] = asunto

email.attach(MIMEText(mensaje, "plain"))

try:

# Conectar al servidor y enviar el correo

with smtplib.SMTP(servidor, puerto) as smtp:

smtp.starttls() # Encriptar conexión

smtp.login(remitente, contraseña)

smtp.sendmail(remitente, destinatario, email.as\_string())

return True

except Exception as e:

print(f"Error al enviar el correo: {e}")

return False

**Paso 3: Integrar el correo en los checklists**

Agrega la lógica para enviar el correo al final de cada checklist (Ingreso, Egreso, Movimiento Interno). Aquí tienes un ejemplo para el **Checklist de Ingreso**:

python

# Al guardar los datos con éxito, envía un correo de notificación

mensaje = f"""

Hola,

Se ha registrado una nueva solicitud de ingreso:

- Solicitante: {solicitante}

- Fecha de ingreso: {fecha\_ingreso}

- Cliente: {cliente}

- DataCenter: {datacenter}

- Ticket: {ticket}

- Cantidad de equipos: {cantidad\_equipos}

¡Saludos!

"""

if enviar\_correo("destinatario@correo.com", "Nueva solicitud de ingreso", mensaje):

st.success("Correo de notificación enviado correctamente.")

else:

st.error("Error al enviar el correo de notificación.")

Haz algo similar para los otros checklists (Egreso y Movimiento Interno), ajustando el contenido del mensaje.

**Paso 4: Probar la funcionalidad**

1. Ejecuta la aplicación y registra una nueva solicitud en cualquier checklist.
2. Verifica que el correo se envíe correctamente al destinatario configurado.
3. Si el correo no se envía, revisa el terminal para verificar si hubo un error (como una contraseña incorrecta o un problema de conexión).

**Paso 5: Personalizar el correo**

Si deseas, puedes personalizar el correo con un formato HTML o agregar más detalles. Cambia MIMEText(mensaje, "plain") por MIMEText(mensaje, "html") y usa HTML para dar formato al cuerpo del correo.

**Siguientes pasos**

Una vez que configuremos correctamente las alertas por correo, podemos planear nuevos pasos:

* Agregar más automatizaciones.
* Refinar gráficos o exportaciones.
* Ampliar la gestión de datos.

Dime cómo te fue con este paso o si prefieres que afinemos algo más antes de seguir avanzando. ¡Estamos logrando grandes cosas, Roberto! 😊🚀

File "c:\Users\Rober\OneDrive\Escritorio\Checklist\_DataCenter\app.py", line 190 elif tipo\_checklist == "Egreso": ^^^^ SyntaxError: invalid syntax

Ese error de SyntaxError significa que algo en la línea previa al bloque elif (o incluso en la misma línea) no está correctamente estructurado en el archivo app.py. Vamos a resolverlo paso a paso:

**1. Identificar posibles causas del error**

El problema suele estar relacionado con:

* Una instrucción previa mal cerrada (por ejemplo, un bloque if sin los dos puntos : o con indentación incorrecta).
* Código mal indentado.
* Un bloque incompleto (por ejemplo, un if sin bloque de código después).
* Algún paréntesis, corchete o comilla sin cerrar antes de esa línea.

**2. Pasos para solucionar**

1. **Revisar la línea previa al error**:
   * Abre tu archivo app.py y revisa la línea inmediatamente antes de:

python

elif tipo\_checklist == "Egreso":

* + Verifica que tenga los dos puntos : si es un bloque if y que esté bien indentada.

1. **Confirmar que las secciones previas están completas**:
   * Por ejemplo, si el error ocurre justo después del bloque de **Movimiento Interno**, asegúrate de que ese bloque esté correctamente cerrado.
   * El bloque debería terminar con las mismas indentaciones, y no debe faltar un return, st.error, etc.
2. **Buscar paréntesis o comillas sin cerrar**:
   * Revisa cualquier línea anterior que incluya paréntesis () o comillas "", '' y asegúrate de que estén cerrados correctamente.

**3. Verifica el archivo de ejemplo**

Aquí tienes una referencia rápida de cómo debería lucir la estructura previa al bloque elif tipo\_checklist == "Egreso"::

python

elif tipo\_checklist == "Ingreso":

st.subheader("Checklist de ingreso")

# Código completo del bloque de Ingreso...

if st.button("Guardar solicitud de ingreso"):

# Código para guardar ingreso...

st.success("¡Solicitud guardada!")

elif tipo\_checklist == "Movimiento interno":

st.subheader("Checklist de movimiento interno")

# Código completo del bloque de Movimiento Interno...

if st.button("Guardar solicitud de movimiento interno"):

# Código para guardar movimiento interno...

st.success("¡Movimiento guardado correctamente!")

elif tipo\_checklist == "Egreso":

st.subheader("Checklist de egreso")

# Aquí comienza el bloque de Egreso...

Si encuentras algo incompleto antes del bloque elif, arréglalo y guarda el archivo.

**4. Probar nuevamente**

Después de corregir el archivo, vuelve a ejecutar el código:

bash

streamlit run app.py

Si el error persiste, comparte las líneas previas al error (alrededor de la línea 190) para que pueda ayudarte a depurarlo con precisión. ¡Vamos a resolverlo juntos! 😊🚀

en esta secuencia esta el error. # Lógica para el checklist de ingreso if tipo\_checklist == "Ingreso": st.subheader("Checklist de ingreso") # Datos básicos del solicitante solicitante = st.text\_input("Nombre del solicitante:") fecha\_ingreso = st.date\_input("Fecha de ingreso:") cliente = st.selectbox("Seleccione el cliente:", obtener\_clientes()) datacenter = st.selectbox("Seleccione el DataCenter:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"]) ticket = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):") # Cantidad de equipos a ingresar cantidad\_equipos = st.number\_input("Cantidad de equipos a ingresar:", min\_value=1, step=1) # Campos dinámicos para cada equipo equipos = [] for i in range(1, cantidad\_equipos + 1): st.subheader(f"Equipo {i}") # Buscador de marcas marca\_busqueda = st.text\_input(f"Buscar marca (Equipo {i}):", key=f"marca\_busqueda\_{i}") marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()] marca\_seleccionada = st.selectbox(f"Seleccione la marca del equipo {i}:", marcas, key=f"marca\_{i}") # Buscador de modelos modelo\_busqueda = st.text\_input(f"Buscar modelo (Equipo {i}):", key=f"modelo\_busqueda\_{i}") modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()] modelo\_seleccionado = st.selectbox(f"Seleccione el modelo del equipo {i}:", modelos, key=f"modelo\_{i}") # Número de serie numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}") equipos.append({"marca": marca\_seleccionada, "modelo": modelo\_seleccionado, "serie": numero\_serie}) # Validación de duplicados numeros\_serie = [equipo["serie"] for equipo in equipos] if len(numeros\_serie) != len(set(numeros\_serie)): st.error("Hay números de serie duplicados en los equipos. Verifique e intente nuevamente.") # Validación de campos obligatorios if not solicitante or not fecha\_ingreso or not cliente or not datacenter or not ticket: st.error("Todos los campos obligatorios deben ser completados.") # Botón para guardar la solicitud if st.button("Guardar solicitud de ingreso"): if all([solicitante, cliente, datacenter, ticket]) and all(e["serie"] for e in equipos): conn = conectar\_db() try: cursor = conn.cursor() for equipo in equipos: cursor.execute(""" INSERT INTO solicitudes\_ingreso ( solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos, marca, modelo, numero\_serie, tipo\_equipo ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?) """, (solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos, equipo["marca"], equipo["modelo"], equipo["serie"], "Automático")) conn.commit() st.success("¡Solicitud de ingreso guardada correctamente!") finally: conn.close() else: st.error("Por favor, complete todos los campos obligatorios.") # Al guardar los datos con éxito, envía un correo de notificación mensaje = f""" Hola, Se ha registrado una nueva solicitud de ingreso: - Solicitante: {solicitante} - Fecha de ingreso: {fecha\_ingreso} - Cliente: {cliente} - DataCenter: {datacenter} - Ticket: {ticket} - Cantidad de equipos: {cantidad\_equipos} ¡Saludos! """ if enviar\_correo("destinatario@correo.com", "Nueva solicitud de ingreso", mensaje): st.success("Correo de notificación enviado correctamente.") else: st.error("Error al enviar el correo de notificación.") # Lógica para el checklist de egreso elif tipo\_checklist == "Egreso": st.subheader("Checklist de egreso") # Datos básicos del solicitante solicitante = st.text\_input("Nombre del solicitante:") fecha\_egreso = st.date\_input("Fecha de egreso:") datacenter\_egreso = st.selectbox("Seleccione el DataCenter de egreso:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"]) ticket\_egreso = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):") # Cantidad de equipos a egresar cantidad\_equipos = st.number\_input("Cantidad de equipos a egresar:", min\_value=1, step=1) # Campos dinámicos para cada equipo equipos = [] for i in range(1, cantidad\_equipos + 1): st.subheader(f"Equipo {i}")

Gracias por compartir la secuencia, Roberto. El error probablemente se debe a la **indentación** o la estructura incompleta en alguna sección del código. Veamos cómo solucionarlo paso a paso:

**1. Causa probable del error**

El problema parece estar en la **indentación** de estas líneas:

python

# Validación de duplicados

numeros\_serie = [equipo["serie"] for equipo in equipos]

if len(numeros\_serie) != len(set(numeros\_serie)):

st.error("Hay números de serie duplicados en los equipos. Verifique e intente nuevamente.")

**Problema:**

* Este bloque está mal indentado, porque no está dentro del flujo de ejecución del bloque if tipo\_checklist == "Ingreso".
* La validación de duplicados y la validación de campos obligatorios deben estar dentro del mismo nivel de indentación que los bloques anteriores del código.

**2. Solución del problema**

Corrige la indentación de las líneas mencionadas para que queden dentro del bloque principal de if tipo\_checklist == "Ingreso":. Aquí tienes la parte corregida:

python

if tipo\_checklist == "Ingreso":

st.subheader("Checklist de ingreso")

# Datos básicos del solicitante

solicitante = st.text\_input("Nombre del solicitante:")

fecha\_ingreso = st.date\_input("Fecha de ingreso:")

cliente = st.selectbox("Seleccione el cliente:", obtener\_clientes())

datacenter = st.selectbox("Seleccione el DataCenter:", ["Providencia", "San Bernardo", "Ascentys", "Bodega"])

ticket = st.text\_input("Ticket asociado (o 'Pendiente' si no existe):")

# Cantidad de equipos a ingresar

cantidad\_equipos = st.number\_input("Cantidad de equipos a ingresar:", min\_value=1, step=1)

# Campos dinámicos para cada equipo

equipos = []

for i in range(1, cantidad\_equipos + 1):

st.subheader(f"Equipo {i}")

# Buscador de marcas

marca\_busqueda = st.text\_input(f"Buscar marca (Equipo {i}):", key=f"marca\_busqueda\_{i}")

marcas = [marca for marca in obtener\_marcas() if marca\_busqueda.lower() in marca.lower()]

marca\_seleccionada = st.selectbox(f"Seleccione la marca del equipo {i}:", marcas, key=f"marca\_{i}")

# Buscador de modelos

modelo\_busqueda = st.text\_input(f"Buscar modelo (Equipo {i}):", key=f"modelo\_busqueda\_{i}")

modelos = [modelo for modelo in obtener\_modelos(marca\_seleccionada) if modelo\_busqueda.lower() in modelo.lower()]

modelo\_seleccionado = st.selectbox(f"Seleccione el modelo del equipo {i}:", modelos, key=f"modelo\_{i}")

# Número de serie

numero\_serie = st.text\_input(f"Número de serie del equipo {i}:", key=f"serie\_{i}")

equipos.append({"marca": marca\_seleccionada, "modelo": modelo\_seleccionado, "serie": numero\_serie})

# Validación de duplicados

numeros\_serie = [equipo["serie"] for equipo in equipos]

if len(numeros\_serie) != len(set(numeros\_serie)):

st.error("Hay números de serie duplicados en los equipos. Verifique e intente nuevamente.")

# Validación de campos obligatorios

if not solicitante or not fecha\_ingreso or not cliente or not datacenter or not ticket:

st.error("Todos los campos obligatorios deben ser completados.")

# Botón para guardar la solicitud

if st.button("Guardar solicitud de ingreso"):

if all([solicitante, cliente, datacenter, ticket]) and all(e["serie"] for e in equipos):

conn = conectar\_db()

try:

cursor = conn.cursor()

for equipo in equipos:

cursor.execute("""

INSERT INTO solicitudes\_ingreso (

solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

marca, modelo, numero\_serie, tipo\_equipo

) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""", (solicitante, fecha\_ingreso, cliente, datacenter, ticket, cantidad\_equipos,

equipo["marca"], equipo["modelo"], equipo["serie"], "Automático"))

conn.commit()

st.success("¡Solicitud de ingreso guardada correctamente!")

finally:

conn.close()

else:

st.error("Por favor, complete todos los campos obligatorios.")

# Al guardar los datos con éxito, envía un correo de notificación

mensaje = f"""

Hola,

Se ha registrado una nueva solicitud de ingreso:

- Solicitante: {solicitante}

- Fecha de ingreso: {fecha\_ingreso}

- Cliente: {cliente}

- DataCenter: {datacenter}

- Ticket: {ticket}

- Cantidad de equipos: {cantidad\_equipos}

¡Saludos!

"""

if enviar\_correo("destinatario@correo.com", "Nueva solicitud de ingreso", mensaje):

st.success("Correo de notificación enviado correctamente.")

else:

st.error("Error al enviar el correo de notificación.")

**3. Probar nuevamente**

1. Guarda el archivo app.py.
2. Ejecuta el proyecto:

bash

streamlit run app.py

1. Verifica que el error desaparezca y que funcione correctamente el checklist de ingreso.

Si el problema persiste, revisa si hay líneas incompletas o mal indentadas en otros bloques del archivo (como el checklist de egreso). ¡Avísame y seguimos ajustándolo juntos! 😊🚀