```
import tkinter as tk
from tkinter import messagebox, filedialog
from tkinter.ttk import Combobox
import snap7
import json
import threading
import time
from datetime import datetime


class BackupApp:
    def __init__(self, root):
        # Initialisation de l'application de sauvegarde
        self.root = root
        self.root.title("Automate Backup")

        # Création des widgets de l'interface utilisateur
        self.create_widgets()

        # Variables pour le thread de sauvegarde et l'état de fonctionnement
        self.backup_thread = None
        self.running = False
        self.save_directory = ""  # Variable pour stocker le répertoire de
sauvegarde sélectionné

    def create_widgets(self):
        # Champ de saisie pour le nom de l'automate
        tk.Label(self.root, text="Nom de l'automate:").grid(row=0, column=0)
        self.name_entry = tk.Entry(self.root)
        self.name_entry.grid(row=0, column=1)

        # Champ de saisie pour l'adresse IP de l'automate
        tk.Label(self.root, text="Adresse IP de l'automate:").grid(row=1,
column=0)
        self.ip_entry = tk.Entry(self.root)
        self.ip_entry.grid(row=1, column=1)

        # Champ de saisie pour le port de l'automate
        tk.Label(self.root, text="Port:").grid(row=2, column=0)
        self.port_label = tk.Label(self.root, text="Port:")
        self.port_label.grid(row=2, column=0)
        self.port_entry = tk.Entry(self.root)
        self.port_entry.grid(row=2, column=1)

        # Liste déroulante pour choisir le type de connexion
        tk.Label(self.root, text="Type de Connexion:").grid(row=3, column=0)
        self.connection_type = Combobox(self.root, values=["Aucun", "Modbus
TCP", "Modbus RTU", "Profinet", "Profibus",
                                                           "Ethernet"])
```

```
        self.connection_type.grid(row=3, column=1)
        self.connection_type.current(0)  # Sélectionne "Aucun" par défaut
        self.connection_type.bind("<<ComboboxSelected>>",
self.on_connection_type_change)

        # Champ de saisie pour le numéro de rack
        self.rack_label = tk.Label(self.root, text="Numéro du rack:")
        self.rack_entry = tk.Entry(self.root)

        # Champ de saisie pour le numéro de slot
        self.slot_label = tk.Label(self.root, text="Numéro du slot:")
        self.slot_entry = tk.Entry(self.root)

        # Liste déroulante pour choisir le format de données à sauvegarder
        tk.Label(self.root, text="Format de Sauvegarde:").grid(row=4,
column=0)
        self.data_format = Combobox(self.root, values=["Programmes",
"Configurations", "Journaux d'événements"])
        self.data_format.grid(row=4, column=1)
        self.data_format.current(0)  # Sélectionne "Programmes" par défaut

        # Bouton pour choisir le répertoire de sauvegarde
        self.dir_button = tk.Button(self.root, text="Choisir répertoire de
sauvegarde", command=self.choose_directory)
        self.dir_button.grid(row=5, column=0, columnspan=2)

        # Bouton pour démarrer la sauvegarde
        self.start_button = tk.Button(self.root, text="Démarrer Sauvegarde",
command=self.start_backup)
        self.start_button.grid(row=6, column=0, columnspan=2)

        # Bouton pour arrêter la sauvegarde
        self.stop_button = tk.Button(self.root, text="Arrêter Sauvegarde",
command=self.stop_backup)
        self.stop_button.grid(row=7, column=0, columnspan=2)
        self.stop_button.config(state=tk.DISABLED)  # Désactiver le bouton
d'arrêt au démarrage

        # Masquer les champs "Numéro du rack" et "Numéro du slot" initialement
        self.rack_label.grid_remove()
        self.rack_entry.grid_remove()
        self.slot_label.grid_remove()
        self.slot_entry.grid_remove()

    def on_connection_type_change(self, event):
        # Désactiver le champ de saisie du port si "Ethernet" est sélectionné
        connection_type = self.connection_type.get()
        if connection_type == "Ethernet":
            self.port_label.grid_remove()
```

```
            self.port_entry.grid_remove()
        else:
            self.port_label.grid(row=2, column=0)
            self.port_entry.grid(row=2, column=1)

        # Afficher les champs "Numéro du rack" et "Numéro du slot" si
"Profinet" ou "Profibus" est sélectionné
        if connection_type in ["Profinet", "Profibus"]:
            self.rack_label.grid(row=5, column=0)
            self.rack_entry.grid(row=5, column=1)
            self.slot_label.grid(row=6, column=0)
            self.slot_entry.grid(row=6, column=1)
            self.dir_button.grid(row=7, column=0,columnspan=2)
            self.start_button.grid(row=8,column=0,columnspan=2)
            self.stop_button.grid(row=9, column=0, columnspan=2)
        else:
            self.rack_label.grid_remove()
            self.rack_entry.grid_remove()
            self.slot_label.grid_remove()
            self.slot_entry.grid_remove()
        if connection_type == "Profibus":
            self.port_entry.config(state=tk.DISABLED)
        else:
            self.port_entry.config(state=tk.NORMAL)

    def choose_directory(self):
        # Ouvre une boîte de dialogue pour sélectionner un répertoire de
sauvegarde
        self.save_directory = filedialog.askdirectory()
        if self.save_directory:
             # Affiche un message informant l'utilisateur du répertoire
sélectionné
            messagebox.showinfo("Répertoire sélectionné",
                                f"Les sauvegardes seront enregistrées dans :
{self.save_directory}")
        else:
            # Affiche un avertissement si aucun répertoire n'a été sélectionné
            messagebox.showwarning("Avertissement",
                                   "Aucun répertoire sélectionné. Les
sauvegardes ne pourront pas être effectuées.")

    def start_backup(self):
         # Récupère les informations saisies par l'utilisateur
        name = self.name_entry.get().strip()
        ip = self.ip_entry.get().strip()
        port = self.port_entry.get().strip()
        rack = self.rack_entry.get().strip()
        slot = self.slot_entry.get().strip()
        connection_type = self.connection_type.get()
```

```
        data_format = self.data_format.get()

        # Vérification des champs
        if not name or not ip or (connection_type not in ["Profibus",
"Profinet"] and not port):
            messagebox.showerror("Erreur", "Tous les champs doivent être
renseignés.")
            return
        if connection_type not in ["Profibus", "Profinet"] and not
port.isdigit():
            messagebox.showerror("Erreur", "Le port doit être un nombre
entier.")
            return
        """
        if connection_type in ["Profibus", "Profinet"] and (not rack.isdigit()
or not slot.isdigit()):
            messagebox.showerror("Erreur", "Le rack et le slot doivent être
des nombres entiers.")
            return
        """
        if not self.save_directory:
            messagebox.showwarning("Avertissement",
                                   "Veuillez sélectionner un répertoire de
sauvegarde avant de démarrer.")
            return

        if connection_type != "Aucun":
            # Initialise la connexion avec l'automate
            if connection_type == "Modbus TCP":
                self.client = ModbusClient(host=ip, port=int(port))
                if not self.client.open():
                    messagebox.showerror("Erreur", "Connexion à l'automate
échouée")
                    return
            elif connection_type == "Modbus RTU":
                # Initialise la connexion Modbus RTU
                # Remplacez ces lignes par l'initialisation de la connexion
Modbus RTU réelle
                self.client = ModbusClient(host=ip, port=int(port))
                if not self.client.open():
                    messagebox.showerror("Erreur", "Connexion à l'automate
échouée")
                    return
            elif connection_type == "Profinet":
                # Utilise Snap7 pour la connexion Profinet
                self.client = snap7.client.Client()
                try:
                    self.client.connect(ip, int(rack), int(slot))
                except Exception as e:
```

```
                    messagebox.showerror("Erreur", f"Connexion Profinet
échouée: {e}")
                    return
            elif connection_type == "Profibus":
                # Utilise Snap7 pour la connexion Profibus
                self.client = snap7.client.Client()
                try:
                    self.client.connect(ip, int(rack), int(slot))
                except Exception as e:
                    messagebox.showerror("Erreur", f"Connexion Profibus
échouée: {e}")
                    return
            elif connection_type == "Ethernet":
                # Placeholder pour la connexion Ethernet
                # Remplacez ces lignes par l'initialisation de la connexion
Ethernet réelle
                self.client = self.initialize_ethernet_connection(ip,
int(port))
                if not self.client:
                    messagebox.showerror("Erreur", "Connexion Ethernet
échouée")
                    return
            else:
                messagebox.showerror("Erreur", "Type de connexion non pris en
charge.")
                return
        else:
            self.client = None

        # Stocke le nom de l'automate et démarre le thread de sauvegarde
        self.automate_name = name
        self.running = True
        self.backup_thread = threading.Thread(target=self.backup_loop)
        self.backup_thread.start()

        # Désactive le bouton de démarrage et active le bouton d'arrêt
        self.start_button.config(state=tk.DISABLED)
        self.stop_button.config(state=tk.NORMAL)

    def stop_backup(self):
        # Arrête le thread de sauvegarde
        self.running = False
        if self.backup_thread:
            self.backup_thread.join()

        # Ferme la connexion avec l'automate
        if self.client and hasattr(self.client, "disconnect"):
            self.client.disconnect()
```

```
        # Active le bouton de démarrage et désactive le bouton d'arrêt
        self.start_button.config(state=tk.NORMAL)
        self.stop_button.config(state=tk.DISABLED)

    def backup_loop(self):
         # Boucle de sauvegarde qui s'exécute dans un thread séparé
        while self.running:
            data = self.read_automate_data()
            if data:
                # Génère un nom de fichier unique avec la date et l'heure
actuelles
                timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
                filename =
f"backup_{self.automate_name}_{timestamp}_{self.data_format.get().replace(' ',
'_')}.json"
                filepath = f"{self.save_directory}/{filename}"
                # Sauvegarde les données dans le fichier
                self.save_backup(data, filepath)
                print(f"Sauvegarde effectuée : {filepath}")
            # Attendre une heure avant la prochaine sauvegarde
            time.sleep(3600)

    def read_automate_data(self):
        if self.client:
            try:
                # Lis les registres de l'automate pour Modbus
                if isinstance(self.client, ModbusClient):
                    return self.client.read_holding_registers(0, 10)
                # Lis les données de l'automate pour Profinet ou Profibus via
Snap7
                elif isinstance(self.client, snap7.client.Client):
                    data = self.client.db_read(1, 0, 10)  # Exemple de
lecture, ajuster selon votre automate
                    return list(data)
            except:
                # Retourne None en cas d'échec
                 return None
        else:
            # Simule des données pour la connexion "Aucun"
            return {"data": "Simulated data because no connection is used"}

    def save_backup(self, data, filename):
        # Prépare les données à sauvegarder avec le nom de l'automate et
l'horodatage
        backup_data = {
            "name": self.automate_name,
            "timestamp": datetime.now().isoformat(),
            "data_format": self.data_format.get(),
            "data": data
```

```
        }
        # Sauvegarde les données dans un fichier JSON
        with open(filename, "w") as f:
            json.dump(backup_data, f)

    def initialize_ethernet_connection(self, ip, port):
        # Placeholder pour la connexion Ethernet
        # Remplacez par le code réel pour établir une connexion Ethernet
        print(f"Établir une connexion Ethernet à {ip}:{port}")
        return True

if __name__ == "__main__":
    # Crée la fenêtre principale de l'application
    root = tk.Tk()
    # Initialise l'application de sauvegarde
    app = BackupApp(root)
    # Démarre la boucle principale de l'interface graphique
    root.mainloop()
```