import requests  
import json  
import urllib3  
from tabulate import tabulate *# Import tabulate for table formatting*import time  
  
*# Suppress SSL warnings*urllib3.disable\_warnings(urllib3.exceptions.InsecureRequestWarning)  
  
*# Standard variables  
# Standard variables*MIST\_API\_TOKEN = "ghh"  
MIST\_BASE\_URL = "<https://api.mist.com/api/v1>"  
ORG\_ID = "hjjjgjgj"  
*# List of sites and their names*SITES = {  
 "USTEST03 - San Mateo": None, *# Site ID will be fetched dynamically* "USTEST04 - San Mateo": None  
}  
  
*# Target OS version*OS\_VERSION = "0.12.27374" *# Replace with the desired OS version  
  
  
# Function to get all sites*def get\_sites(org\_id, api\_token):  
 url = f"{MIST\_BASE\_URL}/orgs/{org\_id}/sites"  
 headers = {  
 "Authorization": f"Token {api\_token}",  
 "Content-Type": "application/json"  
 }  
 try:  
 response = requests.get(url, headers=headers, verify=False)  
 response.raise\_for\_status()  
 return response.json()  
 except requests.exceptions.RequestException as e:  
 print(f"API request failed while fetching sites: {e}")  
 return None  
  
  
*# Function to get available versions for devices in a site*def get\_available\_versions(site\_id, api\_token, model=None):  
 url = f"{MIST\_BASE\_URL}/sites/{site\_id}/devices/versions"  
 headers = {  
 "Authorization": f"Token {api\_token}",  
 "Content-Type": "application/json"  
 }  
 try:  
 response = requests.get(url, headers=headers, verify=False)  
 response.raise\_for\_status()  
  
 available\_versions = response.json()  
 print("Raw response for available versions:", available\_versions)  
  
 *# Filter available versions by model if specified* if model:  
 available\_versions = [  
 version for version in available\_versions if version.get('model') == model  
 ]  
  
 *# Extract version numbers and sort* if available\_versions:  
 versions = [version['version'] for version in available\_versions]  
 sorted\_versions = sorted(versions, key=lambda v: tuple(map(int, v.split('.'))), reverse=True)  
  
 *# Return the sorted versions* return sorted\_versions  
  
 return []  
 except requests.exceptions.RequestException as e:  
 print(f"API request failed while fetching available versions: {e}")  
 return []  
  
  
*# Function to get device details for a site*def get\_device\_details(site\_id, api\_token):  
 url = f"{MIST\_BASE\_URL}/sites/{site\_id}/stats/devices?type=ap"  
 headers = {  
 "Authorization": f"Token {api\_token}",  
 "Content-Type": "application/json"  
 }  
 try:  
 response = requests.get(url, headers=headers, verify=False)  
 response.raise\_for\_status()  
 return response.json()  
 except requests.exceptions.RequestException as e:  
 print(f"API request failed while fetching device details: {e}")  
 return None  
  
  
*# Function to upgrade AP firmware*def upgrade\_ap\_firmware(site\_id, device\_ids, api\_token, version):  
 if not device\_ids:  
 print("No eligible devices for upgrade. Skipping request.")  
 return False  
  
 url = f"{MIST\_BASE\_URL}/sites/{site\_id}/devices/upgrade"  
 headers = {  
 "Authorization": f"Token {api\_token}",  
 "Content-Type": "application/json"  
 }  
 payload = {  
 "version": version,  
 "device\_ids": device\_ids,  
 "strategy": "canary",  
 "canary\_phases": [5, 25, 50, 100],  
 "max\_failures": [1, 1, 5, 5],  
 "enable\_p2p": True,  
 "p2p\_parallelism": 2,  
 "p2p\_cluster\_size": 10,  
 "reboot": True,  
 "force": True  
 }  
  
 try:  
 response = requests.post(url, headers=headers, json=payload, verify=False)  
 response.raise\_for\_status()  
 print(f"✅ Upgrade triggered successfully for Site ID {site\_id}")  
 return True  
 except requests.exceptions.HTTPError as http\_err:  
 print(f"❌ HTTP Error: {http\_err}")  
 print(f"❌ Response Content: {response.text}")  
 return False  
 except requests.exceptions.RequestException as e:  
 print(f"❌ API request failed while triggering upgrade for Site ID {site\_id}: {e}")  
 return False  
  
  
*# Function to check if all devices are upgraded and connected*def check\_upgrade\_status(site\_id, api\_token, device\_ids):  
 *# Poll devices for status after the upgrade* print("Verifying upgrade status...")  
 for attempt in range(5): *# Retry a few times, increasing the wait time* devices = get\_device\_details(site\_id, api\_token)  
 if devices:  
 all\_upgraded = True  
 for device in devices:  
 device\_id = device.get('id')  
 if device\_id in device\_ids:  
 firmware\_version = device.get('version', 'N/A')  
 ap\_status = device.get('status', 'N/A')  
 if firmware\_version != OS\_VERSION or ap\_status != 'connected':  
 all\_upgraded = False  
 print(f"❌ Device {device\_id} failed to upgrade or is not connected.")  
 if all\_upgraded:  
 print("✅ All devices upgraded successfully and are online.")  
 return True  
 print(f"🔄 Retrying in 3 minutes... (Attempt {attempt + 1}/5)")  
 time.sleep(180) *# Wait 3 minutes before retrying  
  
  
# Main execution*if \_\_name\_\_ == "\_\_main\_\_":  
 print("Fetching all sites...\n")  
 sites = get\_sites(ORG\_ID, MIST\_API\_TOKEN)  
  
 if not sites:  
 print("Failed to fetch sites. Exiting.")  
 exit()  
  
 for site\_name in SITES.keys():  
 target\_site = next((site for site in sites if site["name"].lower() == site\_name.lower()), None)  
 if target\_site:  
 SITES[site\_name] = target\_site["id"]  
 else:  
 print(f"Site '{site\_name}' not found. Skipping.")  
  
 for site\_name, site\_id in SITES.items():  
 if site\_id:  
 print(f"\n===== 📍 Site Name: {site\_name} | Site ID: {site\_id} =====\n")  
  
 *# Get available versions dynamically, specifying model* model\_type = "AP43" *# For your case, specify AP43 model here* available\_versions = get\_available\_versions(site\_id, MIST\_API\_TOKEN, model=model\_type)  
  
 if available\_versions:  
 if len(available\_versions) >= 2:  
 latest\_version = available\_versions[0]  
 latest\_1\_version = available\_versions[1]  
 print(f"✅ Latest-1 firmware version for upgrade: {latest\_1\_version}")  
 OS\_VERSION = latest\_1\_version *# Update OS\_VERSION to Latest-1 version* devices = get\_device\_details(site\_id, MIST\_API\_TOKEN)  
 device\_ids = []  
 table\_data = []  
  
 if devices:  
 for device in devices:  
 device\_name = device.get('name', 'N/A')  
 model = device.get('model', 'N/A')  
 firmware\_version = device.get('version', 'N/A')  
 device\_id = device.get('id', 'N/A')  
 ap\_ip = device.get('ip', 'N/A') *# Get AP IP Address* ap\_status = "🟢 Connected" if device.get('status') == 'connected' else "🔴 Disconnected"  
  
 if firmware\_version == OS\_VERSION:  
 status = "✅ Already on target version"  
 elif firmware\_version > OS\_VERSION:  
 status = "⚠️ Newer than target (Skipping)"  
 else:  
 status = "⬆️ Needs Upgrade"  
 device\_ids.append(device\_id)  
  
 table\_data.append([device\_name, model, ap\_ip, ap\_status, firmware\_version, OS\_VERSION, status])  
  
 *# Print table with AP IP and Status* headers = ["Device Name", "Model", "AP IP", "Status", "Current Firmware", "Target OS Version",  
 "Upgrade Status"]  
 print(tabulate(table\_data, headers=headers, tablefmt="fancy\_grid"))  
  
 if device\_ids:  
 confirm = input(  
 f"\nDo you want to proceed with the OS upgrade for site '{site\_name}'? Type 'yes' to continue: ").strip().lower()  
 if confirm == "yes":  
 print(f"Triggering upgrade for site '{site\_name}' to OS version: {OS\_VERSION}")  
 if upgrade\_ap\_firmware(site\_id, device\_ids, MIST\_API\_TOKEN, OS\_VERSION):  
 if check\_upgrade\_status(site\_id, MIST\_API\_TOKEN, device\_ids):  
 print(f"Upgrade completed successfully for site '{site\_name}'.\n")  
 else:  
 print(f"Upgrade failed for site '{site\_name}'. Exiting.")  
 exit()  
 else:  
 print(f"Upgrade failed to trigger for site '{site\_name}'. Exiting.")  
 exit()  
 else:  
 print(f"Upgrade canceled for site '{site\_name}'.")  
 else:  
 print(f"No devices require an upgrade for site '{site\_name}'.\n")  
 else:  
 print(f"Site ID not found for site '{site\_name}'. Skipping.")