VBA

- VBA
- BB
- asdf

VBA

```
Sub GenerateOrgStructure()
    Dim olApp As Outlook.Application
    Dim objNS As Outlook.Namespace
    Dim xlApp As Excel.Application
    Dim wb As Excel.Workbook
    Dim ws As Excel.Worksheet
    Dim addr As Outlook.AddressList
    Dim entry As Object
    Dim row As Long

    ' Initialize Outlook and Excel
    Set olApp = GetObject(, "Outlook.Application")
    Set objNS = olApp.GetNamespace("MAPI")

    ' Create new Excel workbook
    Set xlApp = CreateObject("Excel.Application")
    xlApp.Visible = True
    Set wb = xlApp.Workbooks.Add
    Set ws = wb.Sheets(1)

    ' Set headers
    ws.Cells(1, 1) = "Employee Name"
    ws.Cells(1, 2) = "Email Address"
    ws.Cells(1, 3) = "Manager Name"
    ws.Cells(1, 4) = "Department"
    ws.Cells(1, 5) = "Title"

    ' Get Global Address List
    Set addr = objNS.AddressLists("Global Address List")

    row = 2
    ' Loop through address entries
    For Each entry In addr.AddressEntries
        If entry.Type = "EX" Then
            On Error Resume Next
            Dim exUser As Outlook.ExchangeUser
            Set exUser = entry.GetExchangeUser

            ' Check if the entry is a valid person
            If Not exUser Is Nothing Then
```

```
            Dim manager As Outlook.ExchangeUser
            Set manager = exUser.GetExchangeUserManager

            ' Include only entries with a manager or the CEO (no manager)
            If Not manager Is Nothing Or exUser.JobTitle = "CEO" Then
                ' Write user details to Excel
                ws.Cells(row, 1) = exUser.Name
                ws.Cells(row, 2) = exUser.PrimarySmtpAddress

                ' Get manager info if available
                If Not manager Is Nothing Then
                    ws.Cells(row, 3) = manager.Name
                Else
                    ws.Cells(row, 3) = "No Manager (CEO)"
                End If

                ws.Cells(row, 4) = exUser.Department
                ws.Cells(row, 5) = exUser.JobTitle

                row = row + 1
            End If
        End If
        On Error GoTo 0
    End If
Next entry

' Format as table
ws.Range("A1").CurrentRegion.Select
xlApp.Selection.Format.AutoFit

' Create a table
ws.ListObjects.Add(xlSrcRange, ws.Range("A1").CurrentRegion, , xlYes).Name = "OrgStructure"

' Save the workbook
wb.SaveAs ThisWorkbook.Path & "\OrgStructure_" & Format(Now, "yyyymmdd") & ".xlsx"

' Cleanup
Set ws = Nothing
Set wb = Nothing
Set xlApp = Nothing
Set addr = Nothing
Set objNS = Nothing
Set olApp = Nothing

MsgBox "Org structure has been generated successfully!", vbInformation
End Sub
```

BB

```
Sub ExportGALToExcelOptimized()
    Dim olApp As Outlook.Application
    Dim olNS As Outlook.Namespace
    Dim olGAL As Outlook.AddressList
    Dim olEntries As Outlook.AddressEntries
    Dim olEntry As Outlook.AddressEntry
    Dim olUser As Outlook.ExchangeUser
    Dim xlApp As Object
    Dim xlWB As Object
    Dim xlWS As Object
    Dim i As Long
    Dim dataArray() As Variant

    ' Initialize Outlook
    Set olApp = Outlook.Application
    Set olNS = olApp.GetNamespace("MAPI")
    Set olGAL = olNS.AddressLists("Global Address List")
    Set olEntries = olGAL.AddressEntries

    ' Initialize Excel
    Set xlApp = CreateObject("Excel.Application")
    xlApp.Visible = True
    Set xlWB = xlApp.Workbooks.Add
    Set xlWS = xlWB.Sheets(1)

    ' Add headers to Excel
    xlWS.Cells(1, 1).Value = "Name"
    xlWS.Cells(1, 2).Value = "Email"
    xlWS.Cells(1, 3).Value = "Job Title"
    xlWS.Cells(1, 4).Value = "Department"
    xlWS.Cells(1, 5).Value = "Manager"

    ' Pre-size the array to hold all entries
    ReDim dataArray(1 To olEntries.Count, 1 To 5)

    ' Loop through GAL entries and populate the array
    i = 1
    For Each olEntry In olEntries
        If olEntry.AddressEntryUserType = olExchangeUserAddressEntry Then
            Set olUser = olEntry.GetExchangeUser
            If Not olUser Is Nothing Then
```

```
            dataArray(i, 1) = olUser.Name
            dataArray(i, 2) = olUser.PrimarySmtpAddress
            dataArray(i, 3) = olUser.JobTitle
            dataArray(i, 4) = olUser.Department
            If Not olUser.GetExchangeUserManager Is Nothing Then
                dataArray(i, 5) = olUser.GetExchangeUserManager.Name
            Else
                dataArray(i, 5) = "No Manager"
            End If
            i = i + 1
        End If
    End If
Next olEntry

' Write the array to Excel in one operation
xlWS.Range("A2").Resize(UBound(dataArray, 1), UBound(dataArray, 2)).Value = dataArray

' Autofit columns
xlWS.Columns("A:E").AutoFit

' Clean up
Set olUser = Nothing
Set olEntry = Nothing
Set olEntries = Nothing
Set olGAL = Nothing
Set olNS = Nothing
Set olApp = Nothing

MsgBox "Export complete!", vbInformation
End Sub
```

```
# Import Outlook COM object
$outlook = New-Object -ComObject Outlook.Application
$namespace = $outlook.GetNamespace("MAPI")
$GAL = $namespace.GetGlobalAddressList()
$entries = $GAL.AddressEntries

# Use parallel processing to process entries
$results = $entries | ForEach-Object -Parallel {
    try {
        $user = $_.GetExchangeUser()
        if ($user) {
            [PSCustomObject]@{
```

```
            Name        = $user.Name
            Email       = $user.PrimarySmtpAddress
            JobTitle    = $user.JobTitle
            Department  = $user.Department
            Manager     = if ($user.GetExchangeUserManager()) {
$user.GetExchangeUserManager().Name } else { "No Manager" }
         }
      }
   } catch {}
} -ThrottleLimit 4  # Adjust the throttle limit based on your system

# Export results to CSV
$results | Export-Csv -Path "$env:USERPROFILE\Desktop\GlobalAddressList.csv" -
NoTypeInformation

Write-Host "Export complete! File saved to Desktop."
```

asdf

1. System Prerequisites
First, install required system packages:

bash
Copy Code
```
# Update system
sudo apt update && sudo apt upgrade -y

# Install required system dependencies
sudo apt install -y \
    python3-pip \
    python3-venv \
    chromium \
    chromium-driver \
    xvfb \
    python3-tk \
    python3-dev \
    scrot \
    firefox-esr \
    geckodriver \
    build-essential \
    libssl-dev \
    libffi-dev

# Install X11 utilities for GUI automation
sudo apt install -y x11-utils xdotool
```
2. Project Setup
bash
Copy Code
```
# Create project directory
mkdir ~/youtube_viewer
cd ~/youtube_viewer

# Create virtual environment
python3 -m venv venv

# Activate virtual environment
source venv/bin/activate

# Install Python dependencies
pip install --upgrade pip
```

```
pip install \
    selenium \
    undetected-chromedriver \
    webdriver-manager \
    requests \
    beautifulsoup4 \
    pyautogui \
    numpy \
    python-xlib \
    opencv-python \
    pillow \
    pyvirtualdisplay
```

3. Updated Script Structure

```
youtube_viewer/
├── config/
│   ├── config.json
│   └── proxy_sources.txt
├── logs/
│   └── youtube_viewer.log
├── proxies/
│   └── working_proxies.txt
├── utils/
│   ├── __init__.py
│   ├── browser_manager.py
│   ├── proxy_manager.py
│   └── interaction_manager.py
└── main.py
```

4. Enhanced Implementation

Here's the improved version with better error handling and Debian-specific optimizations:

python
Execute
Copy Code

```
#!/usr/bin/env python3

import json
import os
import random
import time
import logging
import requests
import threading
import sys
import signal
from datetime import datetime, timedelta
from typing import List, Dict, Optional
```

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import (
    TimeoutException,
    WebDriverException,
    NoSuchElementException
)
from webdriver_manager.chrome import ChromeDriverManager
import undetected_chromedriver as uc
from bs4 import BeautifulSoup
import pyautogui
from pyvirtualdisplay import Display

# Configure logging
os.makedirs('logs', exist_ok=True)
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('logs/youtube_viewer.log'),
        logging.StreamHandler()
    ]
)
logger = logging.getLogger(__name__)

class DisplayManager:
    def __init__(self):
        self.display = None

    def start(self):
        """Start virtual display for headless operation"""
        try:
            self.display = Display(visible=0, size=(1920, 1080))
            self.display.start()
            logger.info("Virtual display started successfully")
        except Exception as e:
            logger.error(f"Failed to start virtual display: {e}")
            sys.exit(1)

    def stop(self):
        """Stop virtual display"""
        if self.display:
            self.display.stop()
```

```
            logger.info("Virtual display stopped")

class ProxyManager:
    def __init__(self, config: dict):
        self.config = config
        self.proxies: List[Dict] = []
        self.proxy_lock = threading.Lock()
        self.proxy_sources_file = "config/proxy_sources.txt"
        self.working_proxies_file = "proxies/working_proxies.txt"
        self.last_proxy_update = datetime.min
        self.setup_directories()

    def setup_directories(self):
        """Ensure required directories exist"""
        os.makedirs('config', exist_ok=True)
        os.makedirs('proxies', exist_ok=True)

    def load_proxy_sources(self) -> List[str]:
        """Load proxy source URLs from file with error handling"""
        try:
            if not os.path.exists(self.proxy_sources_file):
                logger.warning("Proxy sources file not found. Creating default...")
                self.create_default_proxy_sources()

            with open(self.proxy_sources_file, 'r') as f:
                sources = [line.strip() for line in f if line.strip() and not line.startswith('#')]
                if not sources:
                    logger.warning("No proxy sources found in file")
                return sources
        except Exception as e:
            logger.error(f"Error loading proxy sources: {e}")
            return []

    def create_default_proxy_sources(self):
        """Create default proxy sources file"""
        default_sources = [
            "https://raw.githubusercontent.com/TheSpeedX/PROXY-List/master/http.txt",
            "https://raw.githubusercontent.com/ShiftyTR/Proxy-List/master/http.txt",
            "https://raw.githubusercontent.com/clarketm/proxy-list/master/proxy-list-raw.txt",
            "https://raw.githubusercontent.com/monosans/proxy-list/main/proxies/http.txt"
        ]
        try:
            with open(self.proxy_sources_file, 'w') as f:
                for source in default_sources:
                    f.write(f"{source}\n")
        except Exception as e:
```

```
            logger.error(f"Failed to create default proxy sources file: {e}")

    def validate_proxy(self, proxy: str) -> Optional[Dict]:
        """Validate proxy with improved error handling"""
        proxy_dict = {
            "http": f"http://{proxy}",
            "https": f"http://{proxy}"
        }

        try:
            start_time = time.time()
            response = requests.get(
                'https://www.youtube.com',
                proxies=proxy_dict,
                timeout=self.config['proxy_settings']['proxy_timeout'],
                headers={'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36'}
            )

            if response.status_code == 200:
                speed = time.time() - start_time
                return {
                    "address": proxy,
                    "speed": speed,
                    "last_checked": datetime.now(),
                    "score": min(100, int(1 / speed * 50))
                }
        except requests.exceptions.RequestException as e:
            logger.debug(f"Proxy validation failed for {proxy}: {str(e)}")
        return None

class BrowserManager:
    def __init__(self, config: dict):
        self.config = config

    def create_driver(self, proxy: Optional[Dict] = None) -> webdriver.Chrome:
        """Create a new Chrome driver with enhanced anti-detection"""
        options = uc.ChromeOptions()

        if proxy:
            options.add_argument(f'--proxy-server={proxy["address"]}')

        # Enhanced anti-detection measures
        options.add_argument('--disable-blink-features=AutomationControlled')
        options.add_argument('--disable-dev-shm-usage')
        options.add_argument('--no-sandbox')
```

```
        options.add_argument('--disable-gpu')
        options.add_argument('--disable-infobars')
        options.add_argument('--disable-notifications')
        options.add_argument('--disable-popup-blocking')

        # Random window size
        window_sizes = [(1920, 1080), (1366, 768), (1440, 900)]
        window_size = random.choice(window_sizes)
        options.add_argument(f'--window-size={window_size[0]},{window_size[1]}')

        try:
            driver = uc.Chrome(options=options)
            return driver
        except Exception as e:
            logger.error(f"Failed to create driver: {e}")
            raise

class YouTubeViewer:
    def __init__(self, config: dict, proxy_manager: ProxyManager, browser_manager:
BrowserManager):
        self.config = config
        self.proxy_manager = proxy_manager
        self.browser_manager = browser_manager
        self.running = True
        signal.signal(signal.SIGINT, self.signal_handler)
        signal.signal(signal.SIGTERM, self.signal_handler)

    def signal_handler(self, signum, frame):
        """Handle shutdown signals gracefully"""
        logger.info("Shutdown signal received. Cleaning up...")
        self.running = False

    def simulate_human_delay(self):
        """Add random delays between actions"""
        time.sleep(random.uniform(1, 3))

    def perform_random_interaction(self, driver: webdriver.Chrome):
        """Perform random human-like interaction with enhanced error handling"""
        try:
            actions = [
                self.scroll_page,
                self.adjust_volume,
                self.toggle_fullscreen,
                self.like_video,
                self.check_comments
            ]
```

```
            action = random.choice(actions)
            action(driver)

        except Exception as e:
            logger.warning(f"Interaction failed: {e}")

    def watch_video(self, driver: webdriver.Chrome, video_url: str):
        """Watch a single video with improved monitoring and interaction"""
        try:
            driver.get(video_url)

            # Wait for video player
            video_element = WebDriverWait(driver, 20).until(
                EC.presence_of_element_located((By.TAG_NAME, "video"))
            )

            # Random watch duration
            watch_duration = random.uniform(
                self.config['viewing_settings']['min_watch_time'],
                self.config['viewing_settings']['max_watch_time']
            )

            start_time = time.time()
            while time.time() - start_time < watch_duration and self.running:
                # Check if video is playing
                if not self.is_video_playing(driver):
                    logger.warning("Video playback stopped. Attempting to resume...")
                    self.resume_playback(driver)

                # Random interactions
                if random.random() < self.config['viewing_settings']['interaction_probability']:
                    self.perform_random_interaction(driver)

                time.sleep(random.uniform(5, 15))

        except Exception as e:
            logger.error(f"Error watching video {video_url}: {str(e)}")
            raise

    def is_video_playing(self, driver: webdriver.Chrome) -> bool:
        """Check if video is currently playing"""
        try:
            return driver.execute_script("return document.querySelector('video').paused") == False
        except:
            return False
```

```
    def resume_playback(self, driver: webdriver.Chrome):
        """Attempt to resume video playback"""
        try:
            driver.execute_script("document.querySelector('video').play()")
        except:
            logger.warning("Failed to resume video playback")

    def run(self):
        """Main viewing loop with improved error handling and recovery"""
        display_manager = DisplayManager()
        display_manager.start()

        while self.running:
            driver = None
            try:
                # Get fresh proxy
                proxy = self.proxy_manager.get_proxy()
                if not proxy:
                    logger.error("No working proxies available")
                    time.sleep(300)
                    continue

                # Create new driver
                driver = self.browser_manager.create_driver(proxy)

                # Process each playlist
                for playlist_id in self.config['playlists']:
                    if not self.running:
                        break

                    self.process_playlist(driver, playlist_id)

            except Exception as e:
                logger.error(f"Error in main loop: {e}")
            finally:
                if driver:
                    try:
                        driver.quit()
                    except:
                        pass
                time.sleep(random.uniform(10, 30))

        display_manager.stop()

def main():
    # Load configuration
    try:
```

```
        with open('config/config.json', 'r') as f:
            config = json.load(f)
    except FileNotFoundError:
        logger.error("Configuration file not found")
        sys.exit(1)
    except json.JSONDecodeError:
        logger.error("Invalid configuration file format")
        sys.exit(1)

    # Initialize managers
    proxy_manager = ProxyManager(config)
    browser_manager = BrowserManager(config)
    viewer = YouTubeViewer(config, proxy_manager, browser_manager)

    try:
        viewer.run()
    except KeyboardInterrupt:
        logger.info("Program terminated by user")
    except Exception as e:
        logger.error(f"Unexpected error: {e}")
    finally:
        logger.info("Cleanup complete")

if __name__ == "__main__":
    main()
```

5. Configuration Files

Create config/config.json:

```
json
Copy Code
{
    "playlists": [
        "YOUR_PLAYLIST_ID_1",
        "YOUR_PLAYLIST_ID_2"
    ],
    "proxy_settings": {
        "min_proxy_score": 70,
        "max_proxy_age_hours": 2,
        "proxy_timeout": 15,
        "rotation_interval": 300
    },
    "viewing_settings": {
        "min_watch_time": 120,
        "max_watch_time": 300,
        "interaction_probability": 0.7,
        "max_retries": 3
```

```
    },
    "browser_settings": {
        "use_virtual_display": true,
        "window_sizes": [
            [1920, 1080],
            [1366, 768],
            [1440, 900]
        ]
    }
}
```

6. Running the Script

bash

Copy Code

```
# Make script executable
chmod +x main.py

# Run the script
./main.py
```

7. Troubleshooting Common Issues

Display Issues

bash

Copy Code

```
# If you get display errors, try:
export DISPLAY=:0
xhost +local:root
```

ChromeDriver Issues

bash

Copy Code

```
# Remove existing ChromeDriver
sudo apt remove chromium-chromedriver

# Install latest version
wget https://chromedriver.storage.googleapis.com/LATEST_RELEASE
VERSION=$(cat LATEST_RELEASE)
wget https://chromedriver.storage.googleapis.com/$VERSION/chromedriver_linux64.zip
unzip chromedriver_linux64.zip
sudo mv chromedriver /usr/local/bin/
sudo chmod +x /usr/local/bin/chromedriver
```

Permission Issues

bash

Copy Code

```
# Fix permissions for the project directory
chmod -R 755 ~/youtube_viewer
```

8. Monitoring and Maintenance

Check Logs

bash

Copy Code
tail -f logs/youtube_viewer.log
Monitor Resource Usage
bash
Copy Code
# Install htop
sudo apt install htop

Monitor resource usage
htop
Clean Up Temporary Files
bash
Copy Code
Add to crontab
0 0 * * * find ~/youtube_viewer/logs -type f -name "*.log" -mtime +7 -delete
9. Security Considerations
Run behind a VPN
Regularly update proxy sources
Monitor for YouTube's anti-bot measures
Use different user agents
Implement random delays between actions
This implementation includes:

Proper error handling
Resource cleanup
Debian-specific optimizations
Virtual display support
Proxy rotation
Anti-detection measures
Logging and monitoring
Security considerations
Remember to:

Regularly update the script
Monitor YouTube's terms of service
Keep system packages updated
Check logs for issues
Maintain proxy lists
Adjust configuration based on performance

https://www.youtube.com/playlist?list=PLrH-0Gmxu9xCs3dSD0EqoNmOEBdg6x2yO
https://www.youtube.com/playlist?list=PL-9Vk7GGcBK212imC7EEJU9KnuxUNpr3C