WhatsApp Message & Media Automator

Aim of the Project

Sending messages to a large number of groups at the same time through whatsapp is a time taking task , with limitations of not forwarding the message which increases the chance of whatsapp ban and losing precious leads, it also becomes a tedious one. This tool brings forward a solution for the same. This tool is designed to help users efficiently manage and deliver timely communications to various groups without manual intervention. Mainly it works on solving three major problems:

- Sending whatsapp messages with a proper schedule to ensure timely communication.
- Decreasing the probability of losing precious leads due to whatsapp ban.
- Reducing Manual labor of monitoring the whatsapp.

How We Achieved It

Overview

We used the Selenium WebDriver to automate interactions with WhatsApp Web, scheduled messages using Python's `datetime`module, and managing group information and message content through JSON and text files.. Additionally, we incorporated a script execution mechanism using Python's `subprocess`module to handle different user requirements such as sending messages, attachments, or both.

Code Snippets and Explanation

Loading WhatsApp Web

We initiate the Edge WebDriver, navigate to WhatsApp Web, and wait for the user to scan the QR code for authentication.

```
# Specify the path to the Edge WebDriver
service = Service('msedgedriver.exe')  # Adjust the path accordingly

# Initialize Edge options
options = Options()

# Create a new instance of the Edge driver
browser = webdriver.Edge(service=service, options=options)
browser.maximize_window()

# Function to wait for an element to be present in the DOM
def wait_for_element(xpath, timeout=30):
    return WebDriverWait(browser, timeout).until(EC.presence_of_element_located((By.XPATH, xpath)))


# Function to load WhatsApp Web and handle retries if it fails to load
def load_whatsapp_web(retries=3):
    for attempt in range(retries):
        try:
            browser.get('https://web.whatsapp.com')
            # Wait for WhatsApp Web to load and the search box to be present
            wait_for_element('//div[@contenteditable="true" and @data-tab="3"]', timeout=60)
            print("WhatsApp Web loaded successfully.")
            return True
        except Exception as e:
            print(f"Error loading WhatsApp Web (Attempt {attempt + 1}/{retries}): {e}")
            time.sleep(5)  # Wait a bit before retrying
    return False

# Attempt to load WhatsApp Web, exit if unsuccessful
if not load_whatsapp_web():
    print("Failed to load WhatsApp Web after multiple attempts.")
    sys.exit(1)

# Wait for user to scan QR code and log in to WhatsApp Web
input("Please scan the QR code and press Enter to continue...")
```

Reading Schedule and Groups

We read the schedule from a JSON file and group names and messages from text files schedule.json and msg.txt respectively. For adding any more messages one can create a text file and change the json files (msg_schedule.json and attach_schedule.py) accordingly.

```
    # Load the schedule of messages from a JSON file
    with open('schedule.json', 'r', encoding="utf8") as f:
        schedule = json.load(f)

    # Load the list of groups from a text file
    with open('groups.txt', 'r', encoding="utf8") as f:
        groups = [group.strip() for group in f.readlines()]
    print("Groups to message:", groups)
```

Filtering Messages and Scheduling

We filter out non-BMP characters from messages and calculate the delay until the scheduled time.

```
# Function to filter out non-BMP characters from text
def filter_non_bmp(text):
    return ''.join(c for c in text if ord(c) <= 0xFFFF)
```

```
    # Prepare the messages to send by reading from the specified files
    messages = []
    for item in schedule:
        message_file = item['message_file']
        with open(message_file, 'r', encoding="utf8") as mess:
            message = mess.read().strip()
            messages.append(filter_non_bmp(message))
    print("Messages to send:", messages)

    # Schedule and send messages at the specified times
    for i, item in enumerate(schedule):
        schedule_time = item['time']
        schedule_time_dt = datetime.strptime(schedule_time, '%Y-%m-%d %H:%M:%S')

        # Calculate the delay in seconds until the scheduled time
        now = datetime.now()
        if schedule_time_dt < now:
            schedule_time_dt += timedelta(days=1)

        delay = (schedule_time_dt - now).total_seconds()
        print(f"Waiting for {delay} seconds to send message '{messages[i]}'")

     # Wait until the scheduled time to send the message
        if delay > 0:
            time.sleep(delay)
```

Sending Messages

We search for each group, open the chat, and send the scheduled message. (messages are present in msg.txt format)
Note: One can schedule any number of messages at any number of time through the msg_schedule.json and attach_schedule.json file.

```
# Send the message to each group
    for group in groups:
        try:
            print(f"Processing group: {group}")

            # Locate and interact with the search box to find the group
            search_box = wait_for_element('//div[@contenteditable="true" and @data-tab="3"]', timeout=10)
            search_box.clear()
            search_box.send_keys(group)
            search_box.send_keys(Keys.ENTER)

            # Wait for the chat to open and be ready
            chat_header_xpath = f'//span[@title="{group}"]'
            wait_for_element(chat_header_xpath, timeout=10)

            # Find the message box and send the message
            message_box_xpath = '//div[@contenteditable="true" and @data-tab="10"]'
            message_box = wait_for_element(message_box_xpath, timeout=10)
            message_box.click()

            # Type and send the message, handling multi-line messages
            for part in messages[i].split('\n'):
                message_box.send_keys(part)
                message_box.send_keys(Keys.SHIFT, Keys.ENTER)

            message_box.send_keys(Keys.ENTER)

            # Minimal delay before moving to the next group
            time.sleep(1)

        except Exception as e:
            print(f"An error occurred while processing group {group}: {e}")
```

Sending Attachments

We search for each group, open the chat, and send the attachments. (One can change the attachments as required from the msg_schedule.json and attach_schedule.json file)

```
if attachment_path:
    # Click the attachment button
    attachment_button = WebDriverWait(browser, 10).until(
        EC.element_to_be_clickable((By.XPATH, ATTACHMENT_BUTTON_XPATH))
    )
    attachment_button.click()

    time.sleep(0.1)  # Ensure attachment menu is fully loaded

    # Upload the image or file
    image_box = WebDriverWait(browser, 10).until(
        EC.presence_of_element_located((By.XPATH, IMAGE_BOX_XPATH))
    )
    image_box.send_keys(os.path.abspath(attachment_path))

    time.sleep(0.1)  # Ensure the image is uploaded
```

Sending Attachments along with Captions(Messages)

We search for each group, open the chat, and send the attachments and messages through the xpath of the .
(One can change the attachments as required from the schedule.json files and messages from the msg.txt files)

```
time.sleep(0.1)  # Ensure attachment menu is fully loaded

# Upload the image or file
image_box = WebDriverWait(browser, 10).until(
    EC.presence_of_element_located((By.XPATH, IMAGE_BOX_XPATH))
)
image_box.send_keys(os.path.abspath(attachment_path))

time.sleep(0.1)  # Ensure the image is uploaded

# Send the message as a caption
caption_box = WebDriverWait(browser, 10).until(
    EC.presence_of_element_located((By.XPATH, CAPTION_BOX_XPATH))
)

# Split message by non-BMP characters and send each part
for part in filter_non_bmp(message).split('\n'):
    caption_box.send_keys(part)
    caption_box.send_keys(Keys.SHIFT, Keys.ENTER)

caption_box.send_keys(Keys.ENTER)

time.sleep(1)  # Ensure the caption is added

# Click the send button
send_button = WebDriverWait(browser, 10).until(
    EC.element_to_be_clickable((By.XPATH, SEND_BUTTON_XPATH))
)
send_button.click()
```

Closing of browser

Closing the browser when all the messages are sent in the groups .

```
finally:
    # Short delay before closing the browser to ensure all messages are sent
    time.sleep(5)
    browser.quit()
```

Script Execution Menu

We created a menu-driven script execution mechanism to allow users to choose between sending messages, attachments, or both.

```python
import subprocess
# Function to execute a given script using subprocess
def execute_script(script_name):
    try:
        subprocess.run(["python", script_name], check=True)  # Run the script with Python interpreter
    except subprocess.CalledProcessError as e:
        print(f"Error executing {script_name}: {e}")  # Print error message if script execution fails

# Main function to present a menu to the user and execute the selected script
def main():
    # Display the menu options
    print("Select an option:")
    print("1: Send message only")
    print("2: Send attachment only")
    print("3: Send both message and attachment")
    print("4: Invalid choice")

    # Get user input for the menu choice
    choice = input("Enter your choice (1-4): ")

    # Execute the corresponding script based on user choice
    if choice == "1":
        print("Executing main.py...")  # Inform the user about the script being executed
        execute_script("main.py")  # Execute the script for sending messages only
    elif choice == "2":
        print("Executing attachment.py...")  # Inform the user about the script being executed
        execute_script("attachment.py")  # Execute the script for sending attachments only
    elif choice == "3":
        print("Executing attach&msg.py...")  # Inform the user about the script being executed
        execute_script("attach&msg.py")  # Execute the script for sending both messages and attachments
    elif choice == "4":
        print("Invalid choice")  # Inform the user about the invalid choice
    else:
        print("Invalid input. Please enter a number between 1 and 4.")  # Inform the user about the invalid input

# Entry point of the script
if __name__ == "__main__":
    main()  # Call the main function
```

How to Use It

1. **Setup Edge WebDriver**: Download and place `msedgedriver.exe` in the project directory. You can use any other browser just have to install the respective web driver (ensure the version of the driver matches with your browser).

   For eg : For chromedriver, the changes in the code will be

   service = Service('chromedriver.exe') # Adjust the path accordingly

   browser = webdriver.Chrome(service=service, options=options)

   Note: These changes will be made in all the three files attachment.py, main.py and attach&msg.py

2. **Prepare `msg_schedule.json, attach_schedule`**: Create a JSON file with the schedule, specifying the message file and the scheduled time.

   (.json file)

```json
[
    {
        "message_file": "msg.txt",
        "time": "2024-06-13 11:49:15"
    },
    {
        "message_file": "msg.txt",
        "time": "2024-06-13 11:49:30"
    }
]
```

3.Prepare `groups.txt`: Create a text file listing the names of the WhatsApp groups, one per line

```
test Group 2
Test group 3
Group 1
```

4.Prepare Message Files: Create text files for each message to be sent, ensuring they match the filenames in the msg_schedule.json and attach_schedule.json files.

```
heyy! this is Yashasvi.😁
I hope u guys r working well.😁😁
```

5.Run the Script: Run the execute.py file.
Execute the script and follow the prompt to scan the QR code. The script will then wait for the scheduled times and send the messages to the specified groups.
Messages sent in every group have been provided an optimal delay as per the whatsapp data to prevent whatsapp from revoking.

Highlights:

Use the Menu for Different Actions: Execute the menu-driven script only (execute.py) to choose between sending messages, attachments, or both.Other three files are private with only administrator access .

Conclusion

This project addresses the challenges of automating WhatsApp messaging with scheduled deliveries, minimizing the risk of account bans due to policy violations, and reducing the need for manual oversight and labor. By leveraging Selenium WebDriver and Python scripting, it provides a robust and efficient solution for managing communication tasks effectively. Users can confidently automate their WhatsApp messaging operations while ensuring compliance and efficiency in their communication strategies.

Github link: https://github.com/ashasvi/WhatsApp_Media_And_Message_Automator-