VIDEO SEARCH AND UPLOAD BOT

PROJECT OVERVIEW:

- The **Video Search and Upload Bot** is a Python-based automation tool designed to streamline the process of downloading videos from social media platforms (Instagram, TikTok), uploading them to a remote server, and managing local video files. The bot continuously monitors a designated folder on the system for new videos, uploads them to a remote platform using pre-signed URLs, and subsequently deletes the local video file after a successful upload.

INSTALLATION AND SETUP DOCUMENTATION:

The following Python modules are required for your **Video Search and Upload Bot** project:

- **requests** - For making HTTP requests to fetch upload URLs and create posts.
- **aiohttp** - For making asynchronous HTTP requests, particularly to upload video files.
- **asyncio** - For asynchronous programming, allowing the bot to run tasks concurrently.
- **os** - For interacting with the operating system, such as checking and creating directories, and removing files.
- **watchdog** - For monitoring the directory for new video files and triggering actions when files are added.
- **time** - For adding time delays, particularly when waiting between actions in the bot (if needed).

To install these dependencies, you can run:

➢ **pip install requests aiohttp watchdog**

CODE DOCUMENTATON:

```
import os
import asyncio
import aiohttp
import requests
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler

# Replace with your actual Flic-Token
FLIC_TOKEN = "flic_d96f9d4110a92cfd83c170211cea387d066e13e48fa43bf9a0499ef23f4db592"

# API Endpoint for uploading video
UPLOAD_URL_ENDPOINT = "https://api.socialverseapp.com/posts/generate-upload-url"

# Headers for API requests
HEADERS = {"Flic-Token": FLIC_TOKEN, "Content-Type": "application/json"}
```

```python
def get_upload_url():
    try:
        response = requests.get(UPLOAD_URL_ENDPOINT, headers=HEADERS)
        response.raise_for_status()
        data = response.json()  # Parse the JSON response
        print(f"Generated upload URL: {data}")
        return data  # Return the data containing the URL and hash
    except requests.RequestException as e:
        print(f"Error fetching upload URL: {e}")
        return None

async def upload_video(pre_signed_url, video_path):
    async with aiohttp.ClientSession() as session:
        try:
            with open(video_path, 'rb') as video_file:
                async with session.put(pre_signed_url, data=video_file) as response:
                    if response.status == 200:
                        print(f"Video {video_path} uploaded successfully!")
                        return True
                    else:
                        print(f"Error uploading video: {response.status}")
                        return False
        except Exception as e:
            print(f"Exception during upload: {e}")
            return False

def create_post(title, hash_value, category_id=1):
    payload = {
        "title": title,
        "hash": hash_value,
        "is_available_in_public_feed": False,
        "category_id": category_id,
    }
    try:
        response = requests.post("https://api.socialverseapp.com/posts", json=payload, headers=HEADERS)
        response.raise_for_status()
        print(f"Post created successfully: {response.json()}")
        return True
    except requests.RequestException as e:
        print(f"Error creating post: {e}")
        return False

class VideoHandler(FileSystemEventHandler):
    def on_created(self, event):
        if event.src_path.endswith(".mp4"):
            print(f"New video detected: {event.src_path}")
            asyncio.create_task(process_video(event.src_path))
```

```
async def process_video(video_path):
    print(f"Processing video: {os.path.basename(video_path)}")
    upload_data = get_upload_url()
    if upload_data:
        pre_signed_url = upload_data.get("url")
        hash_value = upload_data.get("hash")
        if await upload_video(pre_signed_url, video_path):
            if create_post(title=os.path.basename(video_path), hash_value=hash_value):
                os.remove(video_path)
                print(f"Deleted local file: {video_path}")
            else:
                print("Failed to create post.")
        else:
            print("Failed to upload video.")
    else:
        print("Failed to get upload URL.")

async def main():
    # Ensure the videos directory exists
    if not os.path.exists("./videos"):
        os.makedirs("./videos")
    print("Monitoring directory: ./videos")

    event_handler = VideoHandler()
    observer = Observer()
    observer.schedule(event_handler, path="./videos", recursive=False)
    observer.start()

    print("Bot is running. Press Ctrl+C to stop.")
    try:
        while True:
            await asyncio.sleep(1)  # Keep the bot running
    except KeyboardInterrupt:
        print("KeyboardInterrupt received. Stopping bot...")
        observer.stop()
        observer.join()

# Run the bot using asyncio.run()
if __name__ == "__main__":
    asyncio.run(main())  # This will run the main function asynchronously
```

ERROR HANDLING AND TROUBLESHOOTING:

Error Handling in the Bot:

- **Network Issues**: Errors such as timeouts, connection issues, or invalid responses from the server when requesting an upload URL or uploading a video.

- **Solution**: Use try-except blocks to catch these exceptions. Log the error message to the console and possibly retry the request with a delay.

```
try:
    response = requests.get(UPLOAD_URL_ENDPOINT, headers=HEADERS)
    response.raise_for_status()
except requests.RequestException as e:
    print(f"Error fetching upload URL: {e}")
    return None
```

File Handling Errors: Errors when accessing video files (e.g., file not found or permission issues).

- **Solution**: Use appropriate file path checks and error handling during file operations.

```
try:
    with open(video_path, 'rb') as video_file:
        # upload code here
except Exception as e:
    print(f"Error uploading video: {e}")
```

2. Troubleshooting Common Issues:

- **Bot Not Running**:
    - **Cause**: The bot may not be running if the directory doesn't exist, or if the event handler is misconfigured.
    - **Solution**: Ensure that the ./videos directory exists and is correctly monitored by the watchdog observer.

```
if not os.path.exists("./videos"):
    os.makedirs("./videos")
```

API Request Issues:

- **Cause**: Incorrect API token or endpoint may cause authentication or request failures.
- **Solution**: Double-check the Flic API token and ensure that the API endpoint is correct and reachable.

```
FLIC_TOKEN = "flic_d96f9d4110a92cfd83c170211cea387d066e13e48fa43bf9a0499ef23f4db592"
```

API DOCUMENTATION:

If you're interacting with a third-party API (like the one for uploading videos), you should document the API usage, the required endpoints, and their expected responses.

1. GET /posts/generate-upload-url

- **Purpose**: This API endpoint generates a pre-signed URL for uploading a video.
- **Request**:
    - **Method**: GET
    - **Headers**:
        - Flic-Token: Your unique API token.
    - **Response**: A JSON object with the upload URL and hash.

```
{
  "url": "https://upload-flic.com/signed_url",
  "hash": "video_file_hash_value"
}
```

2. POST /posts

- **Purpose**: This API endpoint creates a post for the uploaded video.
- **Request**:
    - **Method**: POST
    - **Headers**:
        - Flic-Token: Your unique API token.

```
{
  "title": "My Video Title",
  "hash": "video_file_hash_value",
  "is_available_in_public_feed": false,
  "category_id":
}
```

3. Error Handling in API:

- If the request fails (e.g., due to invalid token, network issues, or server errors), the API will respond with an error message. Example error response:

```
{
  "id": "123456",
  "title": "My Video Title",
  "status": "success"
}
```

3. Error Handling in API:

- If the request fails (e.g., due to invalid token, network issues, or server errors), the API will respond with an error message. Example error response:

```
{
   "error": "Invalid API token"
}
```

4. Authentication:

- The API uses a unique **Flic-Token** for authentication.
- Make sure that the token is valid and not expired.

CONCLUSION:

- The **Video Search and Upload Bot** project is designed to automate the process of monitoring a directory for new video files, uploading them to a remote server via API, and then creating a post for each video. The bot makes use of asynchronous programming to handle multiple tasks concurrently, ensuring efficiency and responsiveness while working with external APIs.