Audio & Video Filter Web Application

Streaming Processed Multimedia Content

Rodion Makarov s270470
GitHub

Music and Multimedia streaming over the Internet

2025

Project Overview

[GitHub]

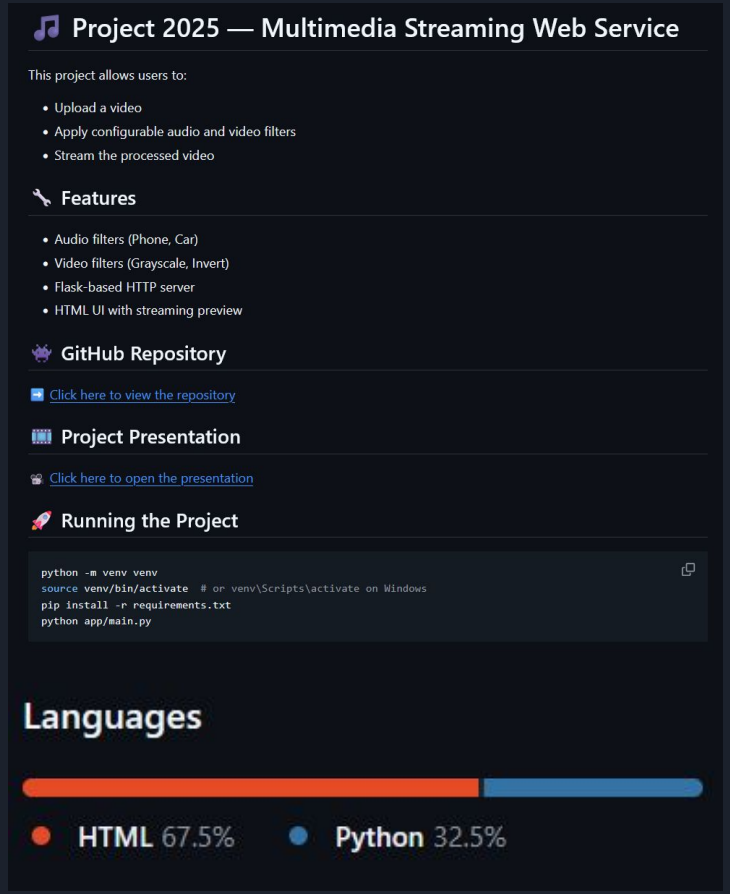
Goal: To apply filters to *MP4* files via a web interface, allowing users to watch the result via streaming

Key Functionality:

1. Upload *MP4* files in the browser.
2. Choose and stack audio and video filters.
3. Configure filter parameters.
4. Stream processed video.

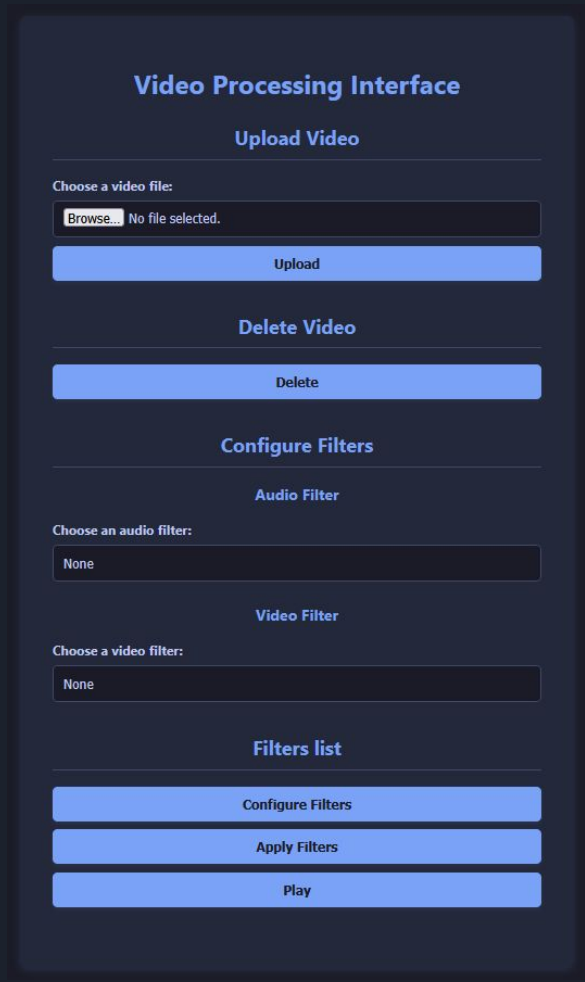
Technologies Used:

- Frontend: HTML + JavaScript
- Backend: Python
- Processing: FFmpeg (for video), NumPy, SciPy (for audio)



Key Features & User Experience

- Clean and simple UI
- File Upload & Management:
  - Upload MP4 files.
  - Option to delete previously uploaded videos.
  - Only one video can be present on the server at a time.
- Configurable Filters:
  - User can select one or more implemented filters for both audio and video.
  - Parameters for filters can be configured.
- Seamless Streaming of Processed Content



Code Architecture

```
├── .git/ Git repository
├── .gitignore   Files that Git should ignore
├── app/ Contains the main application logic
│   ├── main.py Main application file
│   ├── video_filters.py Contains video filters
│   └── audio_filters.py Contains audio filters
└── templates/ Contains HTML templates for the user interface
    └── index.html Main HTML file for the web application
```

Implemented Video Filters

Grayscale:

- Converts the video to black and white.
- Strips chroma channels, preserving only brightness (luminance).
- *Show relevant code snippet (like the one in your PPT for* **grayscale_filter***).*

Invert Colors:

- Inverts each YUV channel in the video.
- *Show relevant code snippet (like the one in your PPT for* **invert_colors***).*

Python

```python
import subprocess

def invert_colors(input, output):
    command = [
        "ffmpeg", "-y", "-i", input,
        "-vf", "lutyuv=y=negval:u=negval:v=negval",
        output
    ]
    try:
        subprocess.run(command, check=True)
        print(f"Inverted video saved to {output}")
    except subprocess.CalledProcessError as e:
        print(f"An error occurred: {e}")
        raise

def grayscale_filter(input, output):
    command = [
        "ffmpeg", "-y", "-i", input,
        "-vf", "format=gray",
        output
    ]
    try:
        subprocess.run(command, check=True)
        print(f"Grayscale video saved to {output}")
    except subprocess.CalledProcessError as e:
        print(f"An error occurred: {e}")
        raise
```

Implemented Audio Filters - part 1

Phone Filter:

- Simulates phone-like audio quality.
- Attenuates stereo side signal (mono effect).
- Applies a bandpass filter in the range 800–12000 Hz using SciPy's butter_bandpass and sosfilt.
- *Show relevant code snippet (like the one in your PPT for* **phone** *filter).*

```
def phone(input_path, output_path, order, gain):
    sample_rate, audio_data = wav.read(input_path)
    audio_data = audio_data.astype(float)

    if audio_data.ndim == 2 and gain == 0:
        left = audio_data[:, 0]
        right = audio_data[:, 1]
        mid = (left + right) / 2
        side = (left - right) / 2 * 0.3
        left = mid + side
        right = mid - side
        sos = butter_bandpass(800, 12000,
sample_rate, order)
        left_filtered = sosfilt(sos, left)
        right_filtered = sosfilt(sos, right)
        filtered = np.vstack([left_filtered,
right_filtered]).T
    else:
        sos = butter_bandpass(800, 12000,
sample_rate, order)
        filtered = sosfilt(sos, audio_data)

    filtered = np.clip(filtered, -32768,
32767).astype(np.int16)
    wav.write(output_path, sample_rate, filtered)
```

Implemented Audio Filters - part 2

Car Filter:

- Simulates audio heard inside a car.
- Amplifies stereo side signal for spatial feel.
- Applies a low-pass filter at 3 kHz to muffle high tones.
- Combines mid/side processing and filtering.
- *Show relevant code snippet (like the one in your PPT for* **car** *filter).*

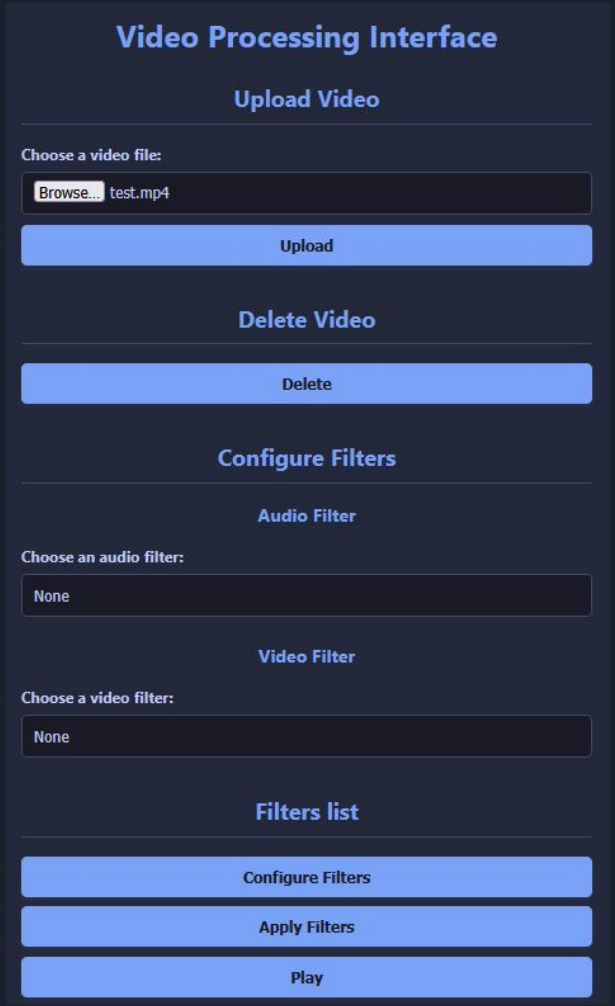
```
def car(input_path, output_path, order, gain):
    sample_rate, audio_data = wav.read(input_path)
    audio_data = audio_data.astype(float)

    if audio_data.ndim == 2:
        left = audio_data[:, 0]
        right = audio_data[:, 1]
        mid = (left + right) / 2
        side = (left - right) / 2 * gain
        left = mid + side
        right = mid - side
        sos = butter(order, 3000, btype='low',
fs=sample_rate, output='sos')
        left_filtered = sosfilt(sos, left)
        right_filtered = sosfilt(sos, right)
        filtered = np.vstack([left_filtered,
right_filtered]).T
    else:
        sos = butter(order, 3000, btype='low',
fs=sample_rate, output='sos')
        filtered = sosfilt(sos, audio_data)

    filtered = np.clip(filtered, -32768,
32767).astype(np.int16)
    wav.write(output_path, sample_rate, filtered)
```

System Workflow

- **Modular and Stackable Filter Architecture.**
- **Step-by-Step Process:**
  1. *Upload Video:* User uploads an MP4 file through the web interface.
  2. *Configure Filters:* User selects desired audio and video filters and their parameters via dropdown menus.
  3. *Apply Filters:* The server processes the uploaded video, applying the configured audio and video filters. This step involves generating a second video with applied filters.
  4. *Stream Result:* Once processing is complete, the user can stream the filtered video.



Endpoint from **main.py** *- part 1*

`@app.route('/')`

Purpose: Serves the main HTML page (index.html) and initializes the application state by ensuring no previous video files remain from past sessions.

```
@app.route('/')
def index():
    for path in [VIDEO_PATH, OUTPUT_PATH]:
        if os.path.exists(path):
            os.remove(path)
    return render_template('index.html')
```

Endpoint from **main.py** - *part 2*

`@app.route('/upload', methods=['POST'])`

Purpose: This endpoint allows clients to upload a video file to the server.

How it works:

- It accepts a POST request at /upload with a file field named 'video'.
- If a video already exists at VIDEO_PATH, it returns an error.
- If no file is provided, it returns an error.
- Otherwise, it saves the uploaded video to VIDEO_PATH and returns a success message.

```python
@app.route('/upload', methods=['POST'])
def upload():
    if os.path.exists(VIDEO_PATH):
        return jsonify({'error': 'A video is already uploaded'}), 400
    if 'video' not in request.files:
        return jsonify({'error': 'No file selected'}), 400
    file = request.files['video']

    file.save(VIDEO_PATH)
    return jsonify({'message': 'Video uploaded successfully'}), 200
```

Endpoint from **main.py** - *part 3*

`@app.route('/delete', methods=['DELETE'])`

Purpose: This endpoint deletes the uploaded video and its processed output from the server.

How it works:

- It accepts a DELETE request at /delete.
- It checks if files exist at VIDEO_PATH and OUTPUT_PATH, and removes them if found.
- If any file was deleted, it returns a success message.
- If no files were found, it returns a "Nothing to delete" message with a 400 status.

```python
@app.route('/delete', methods=['DELETE'])
def delete():
    deleted_any = False
    for path in [VIDEO_PATH, OUTPUT_PATH]:
        if os.path.exists(path):
            os.remove(path)
            deleted_any = True
    if deleted_any:
        return jsonify({'message': 'Video deleted successfully'}), 200
    else:
        return jsonify({'message': 'Nothing to delete'}), 400
```

Endpoint from **main.py** - *part 4*

`@app.route('/configure', methods=['POST'])`

Purpose: This endpoint sets filter configurations for future video processing.

How it works:

- It accepts a POST request at /configure with a JSON body containing a filters list.
- It updates the global FILTERS variable with the provided filters.
- Returns a success message confirming the filters were set.

```python
@app.route('/configure', methods=['POST'])
def configure():
    global FILTERS
    data = request.get_json()
    FILTERS = data.get('filters', [])
    return jsonify({'message': f'Filters set: {FILTERS}'}), 200
```

Endpoint from **main.py** *- part 5*

`@app.route('/apply', methods=['POST'])`

Purpose: This endpoint applies a series of configured audio/video filters to an uploaded video.

How it works:

- Accepts a POST request at `/apply`.
- Checks that a video has been uploaded.
- Creates a temporary copy of the video using `ffmpeg`.
- Iterates over the global `FILTERS` list and applies each filter in sequence:
  - **Video filters**: like `color invert`, `grayscale`.
  - **Audio filters**: like `phone`, `car`, with custom parameters.
- Uses external tools (`ffmpeg`) and custom functions (`invert_colors`, `grayscale_filter`, `phone`, `car`) to process the *video/audio*.
- Replaces the original processed video with the final output.
- Returns success or an error message if any filter fails.

Endpoint from main.py *- part 6*

`@app.route('/stream-video')`

Purpose: This endpoint streams the available video to the client.

How it works:

- Accepts a GET request at /stream-video.
- If a processed video exists at OUTPUT_PATH, it streams that.
- If only the original uploaded video exists at VIDEO_PATH, it streams that instead.
- If no video is found, it returns a 404 error.

```python
@app.route('/stream-video')
def stream_video():
    if os.path.exists(OUTPUT_PATH):
        return send_file(OUTPUT_PATH,
mimetype='video/mp4')
    elif os.path.exists(VIDEO_PATH):
        return send_file(VIDEO_PATH,
mimetype='video/mp4')
    else:
        return abort(404, description="No video
available to stream.")
```

Rodion Makarov s270470 GitHub

18 pls…🥺