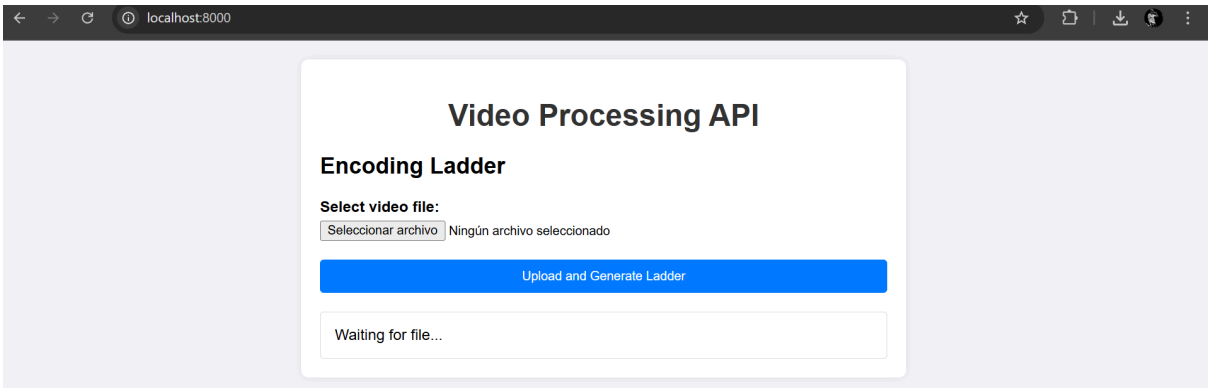
Ex3:

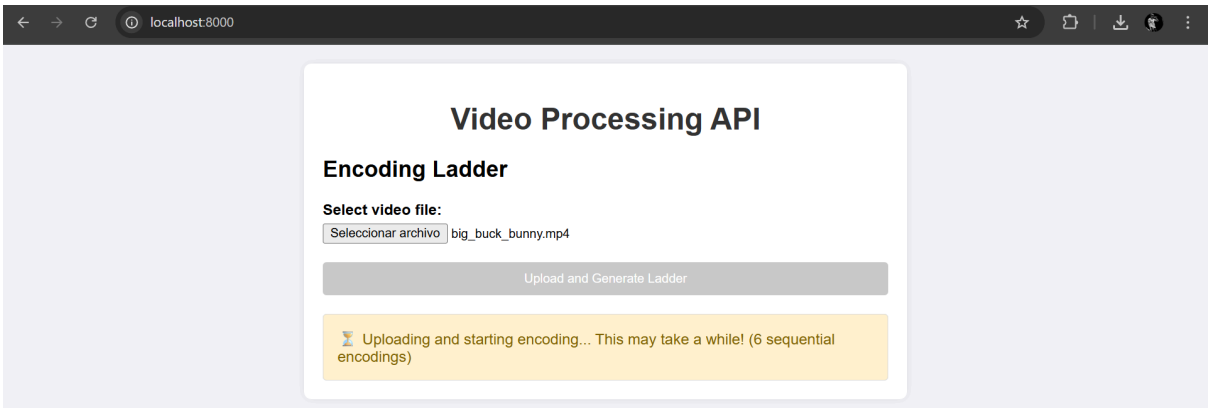
Previous Code:

```HTML
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Video API</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px;
background-color: #f4f4f9; }
        .container { max-width: 600px; margin: auto; background: white;
padding: 20px; border-radius: 8px; box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
        h1 { color: #333; text-align: center; }
        label { display: block; margin-top: 10px; font-weight: bold; }
        input[type="file"] { margin-top: 5px; }
        button { background-color: #007bff; color: white; padding: 10px
15px; border: none; border-radius: 4px; cursor: pointer; margin-top: 20px;
width: 100%; }
        button:disabled { background-color: #ccc; cursor: not-allowed; }
        #status { margin-top: 20px; padding: 15px; border-radius: 4px;
border: 1px solid #ddd; }
        .loading { background-color: #fff3cd; color: #856404; }
        .success { background-color: #d4edda; color: #155724; }
        .error { background-color: #f8d7da; color: #721c24; }
        .output a { display: block; margin-top: 5px; color: #007bff; }
    </style>
</head>
<body>
    <div class="container">
        <h1>Video Processing API</h1>
        <h2>Encoding Ladder</h2>

        <form id="upload-form">
            <label for="video-file">Select video file:</label>
            <input type="file" id="video-file" name="file" accept="video/*"
required>

            <button type="submit" id="submit-btn">
                Upload and Generate Ladder
            </button>
        </form>

        <div id="status" class="status">Waiting for file...</div>
```

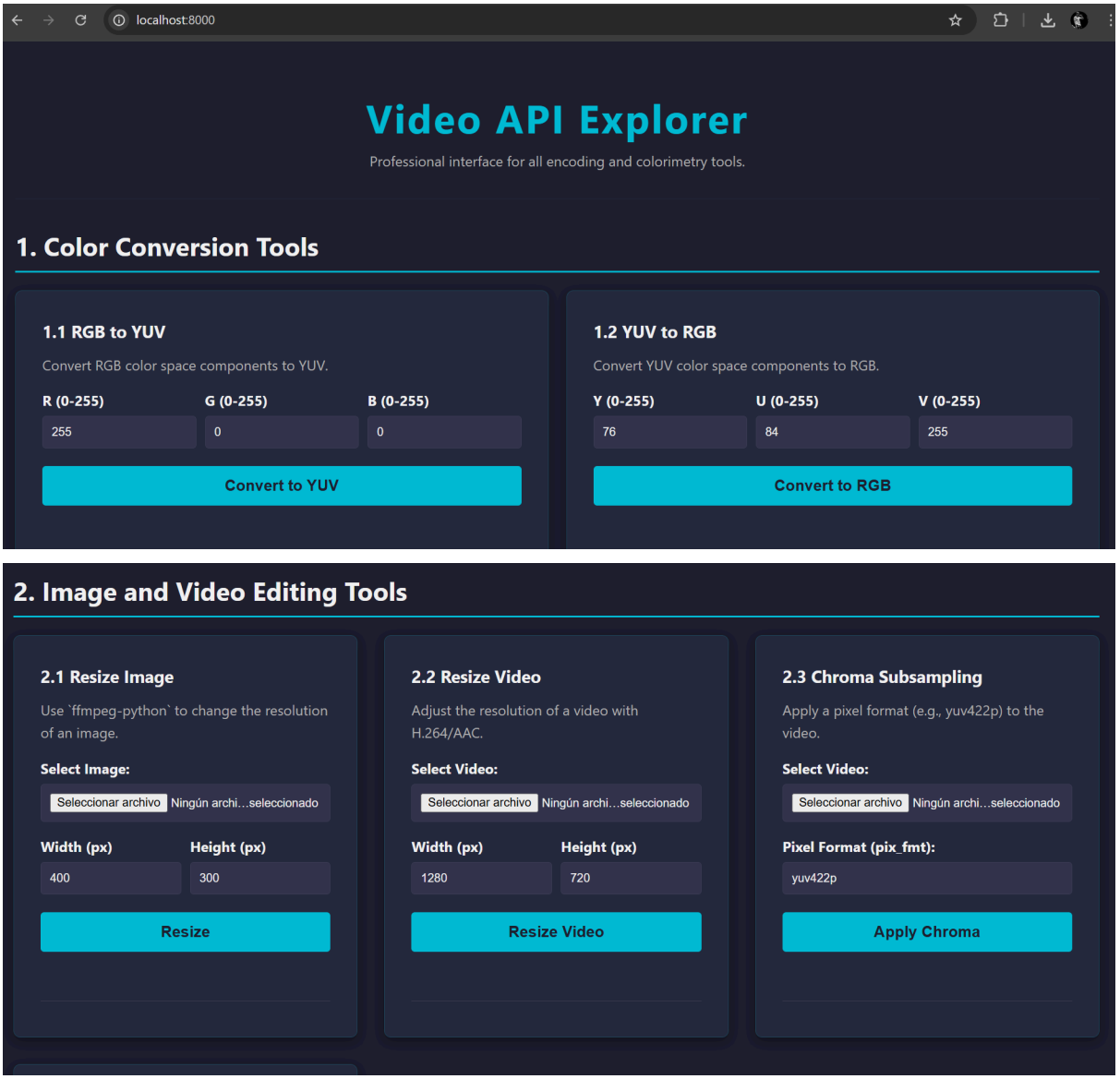
```
        <div id="output" class="output"></div>
    </div>

    <script>
        const form = document.getElementById('upload-form');
        const fileInput = document.getElementById('video-file');
        const statusDiv = document.getElementById('status');
        const outputDiv = document.getElementById('output');
        const submitBtn = document.getElementById('submit-btn');

        const API_BASE_URL = 'http://localhost:8000';

        form.addEventListener('submit', async (e) => {
            e.preventDefault();

            const file = fileInput.files[0];
            if (!file) {
                statusDiv.className = 'status error';
                statusDiv.innerHTML = 'Please, select a video file to
upload.';
                return;
            }

            const formData = new FormData();
            formData.append('file', file);

            statusDiv.className = 'status loading';
            statusDiv.innerHTML = '⏳ Uploading and starting encoding...
This may take a while! (6 sequential encodings)';
            outputDiv.innerHTML = '';
            submitBtn.disabled = true;

            try {
                const response = await
fetch(`${API_BASE_URL}/video/encoding_ladder/`, {
                    method: 'POST',
                    body: formData
                });

                const data = await response.json();
                if (response.ok) {
                    statusDiv.className = 'status success';
                    statusDiv.innerHTML = `✅ ${data.message}`;

                    let htmlOutput = '<h3>Generated files:</h3>';
                    data.ladder.forEach(item => {
                        htmlOutput += `<p>Resolution ${item.resolution}
(${item.bitrate}):</p>`;
```

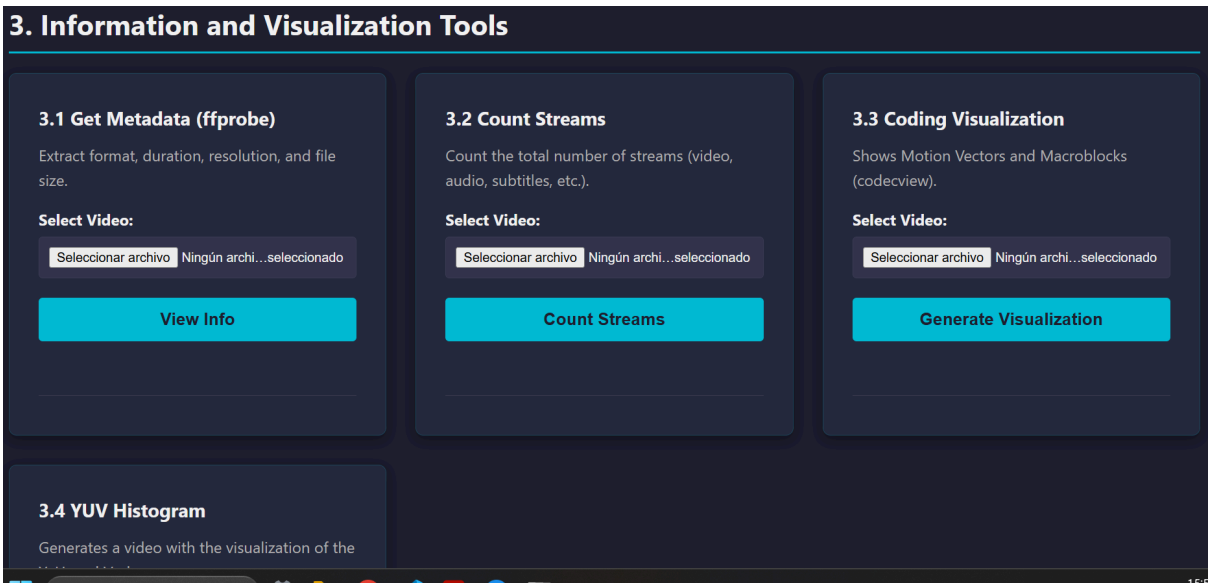
```
                        htmlOutput += `<a href="${item.file}"
target="_blank">${item.file}</a>`;
                    });
                    outputDiv.innerHTML = htmlOutput;

                } else {
                    statusDiv.className = 'status error';
                    statusDiv.innerHTML = `❌ API error: ${data.detail ||
'Unknown error'}`;
                }

            } catch (error) {
                statusDiv.className = 'status error';
                statusDiv.innerHTML = `❌ Network or server error:
${error.message}`;
            } finally {

                submitBtn.disabled = false;
            }
        });
    </script>
</body>
</html>
```

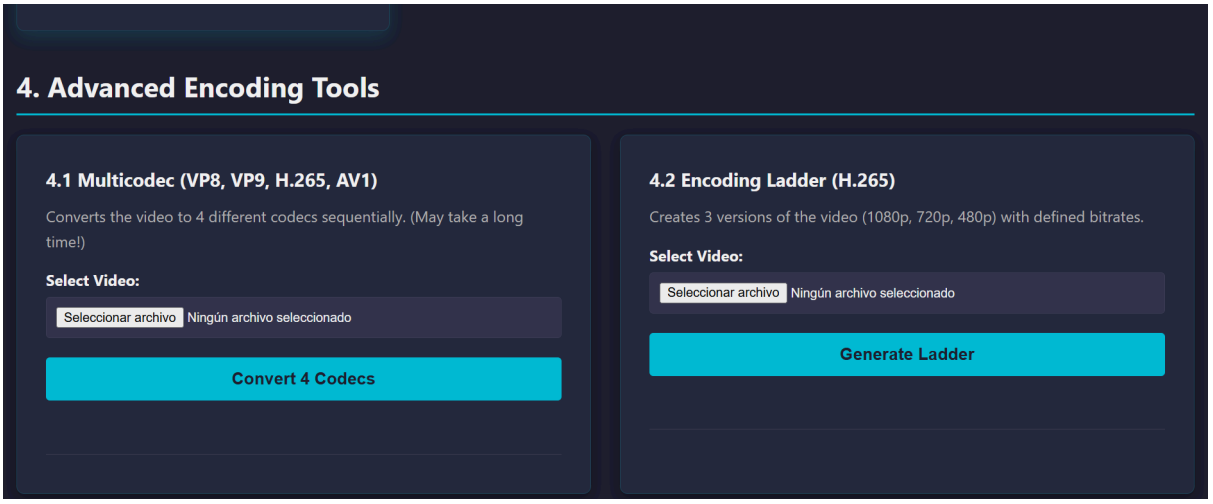




Ex4:


localhost:8000
Video API Explorer
Professional interface for all encoding and colorimetry tools.
1. Color Conversion Tools
1.1 RGB to YUV
Convert RGB color space components to YUV.
R (0-255)
G (0-255)
B (0-255)
255
0
0
Convert to YUV
1.2 YUV to RGB
Convert YUV color space components to RGB.
Y (0-255)
U (0-255)
V (0-255)
76
84
255
Convert to RGB
2. Image and Video Editing Tools
2.1 Resize Image
Use `ffmpeg-python` to change the resolution of an image.
Select Image:
Seleccionar archivo
Ningún archi...seleccionado
Width (px)
Height (px)
400
300
Resize
2.2 Resize Video
Adjust the resolution of a video with H.264/AAC.
Select Video:
Seleccionar archivo
Ningún archi...seleccionado
Width (px)
Height (px)
1280
720
Resize Video
2.3 Chroma Subsampling
Apply a pixel format (e.g., yuv422p) to the video.
Select Video:
Seleccionar archivo
Ningún archi...seleccionado
Pixel Format (pix_fmt):
yuv422p
Apply Chroma





Differences:

Theme and Styling:

- **Exercise 3:** Basic light theme with CSS styles embedded directly within the HTML <style> tag.
- **Final Version:** Professional dark theme using custom CSS variables (like --primary-color: #00bcd4; for the vibrant accent) defined in a separate styles.css file. This drastically improves organization and visual appeal.

Content Structure:

- **Exercise 3:** Linear content structure centered in a single, narrow container.
- **Final Version:** Modular design using cards (.card) and a responsive grid layout (.module-grid). The application adapts well to different screen sizes and logically groups the functionalities into 4 distinct sections (Color, Editing, Info, Advanced).

API Coverage:

- **Exercise 3:** Focused only on the "Encoding Ladder" endpoint (/video/encoding_ladder/).
- **Final Version:** Integrates 12 different functionalities across 4 logical categories, making it a comprehensive API explorer:
    - **Color Conversion:** RGB to YUV and vice versa.
    - **Image & Video Editing:** Resize, Chroma Subsampling, Multi-track Audio.
    - **Information & Visualization:** Metadata (ffprobe), Stream Count, Coding Visualization, and YUV Histogram.
    - **Advanced Encoding:** Multicodec (VP8, VP9, H.265, AV1) and the original Encoding Ladder.

Form Handling Engine:

- **Exercise 3:** The fetch logic was hardcoded and specific to the single file upload form.
- **Final Version:** Uses a single, powerful, and dynamic handleFormSubmit function. This function reads the data-endpoint and data-type attributes from the card to determine:

1. The target API URL.
2. How to construct the request body: as JSON (for color conversions), as FormData with URL Query Parameters (file-query, for resizing), or just as FormData (file).

Output Processing:

- **Exercise 3:** Only knew how to process the data.ladder list.
- **Final Version:** The function dynamically checks the response structure based on the called endpoint. It intelligently displays raw JSON output, structured video metadata lists, single download links, or multiple download links (for the multicodec and ladder results).