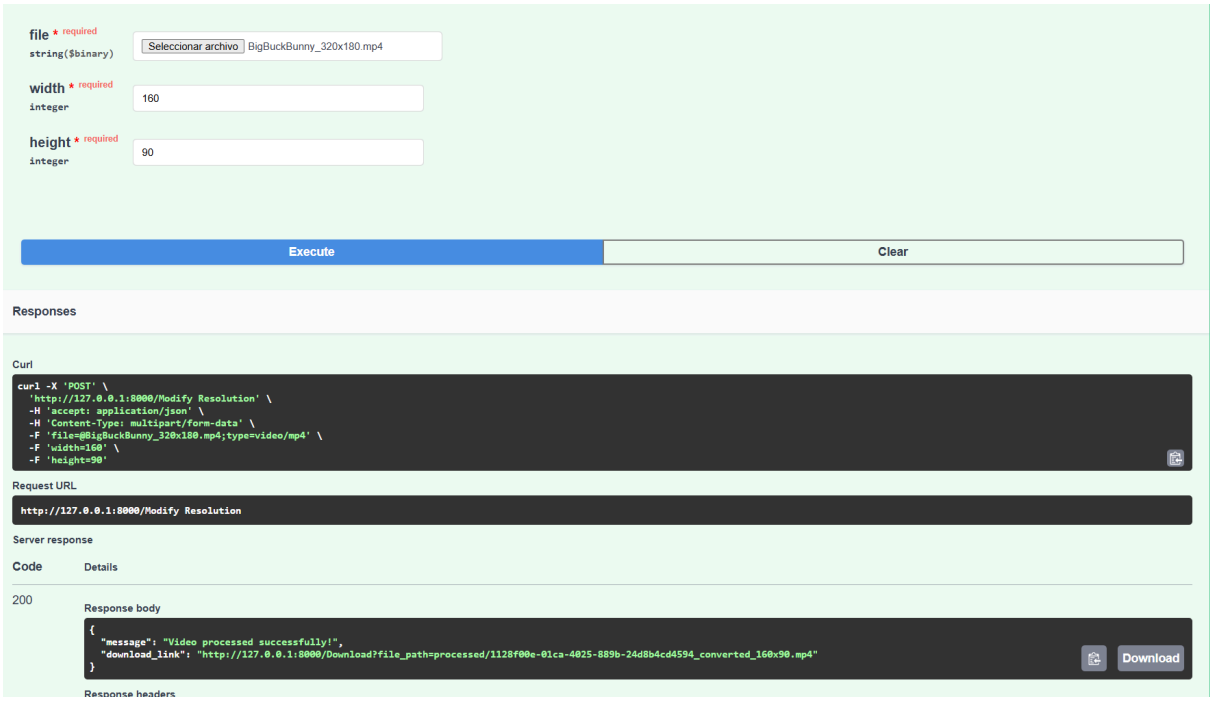
Javier Portabella 251515
Oriol Sánchez 253207

S2: MPEG4 and more endpoints

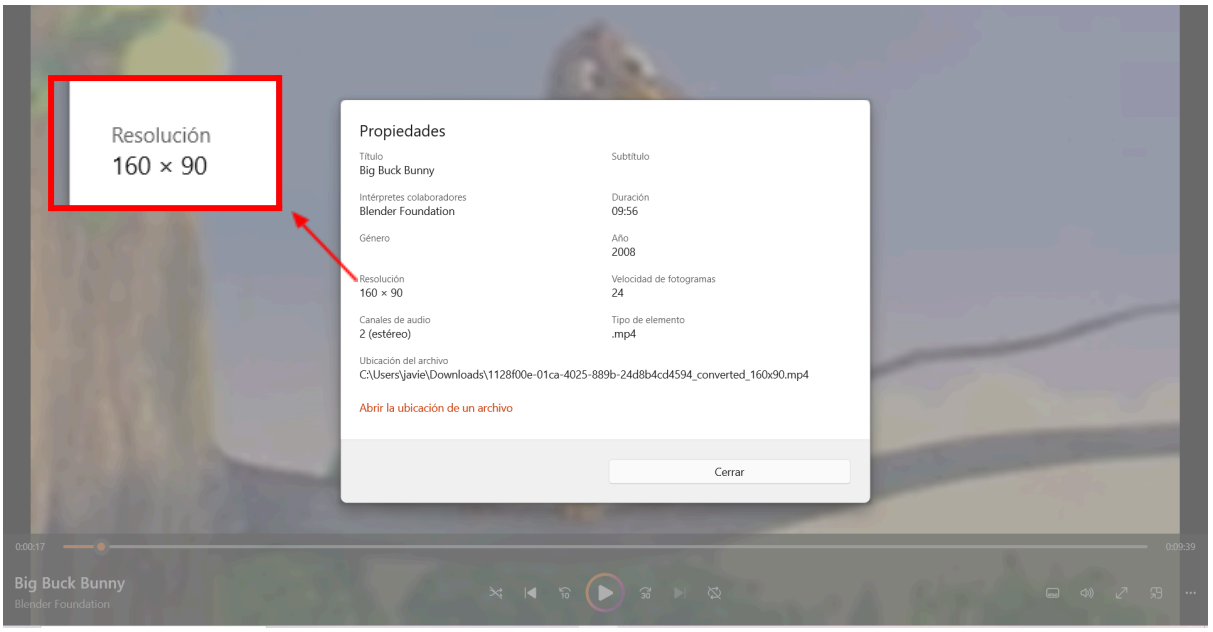
Report

Following the steps, we first downloaded the Big Buck Bunny video. The resolution was 320x180. So the first step we took was to make a new feature where you can modify this resolution.

In every function, there is at the top the explanation of what the function does. At the beginning, when we have done it just using fastapi, the functions returned the video to the preselected output path (in the “processed” folder of the files). However, in order to work the api as a docker, we needed to change this and we decided to provide the link to download the processed files.



We can see in the image how the resolution of the video is being changed to half of the original:



In the second exercise we did not get the expected results and although making many changes, we were not able to get the chroma subsampled version of the video. We finally discovered that the problem was in entering the pixel format correctly. It can only be yuv420p, yuv422p, yuv444p or others but following the same string structure.

The third feature of our API works as expected and returns the duration of the video in seconds, the bitrate in bits per second, the width, the heigh, the codec that was used to encode it and the format.

Response body


```
{
  "message": "Video info extracted",
  "video_info": {
    "Duration (s)": "9:56",
    "Bitrate (bps)": 867212,
    "Width": 320,
    "Height": 180,
    "Codec": "h264",
    "Format": "mov,mp4,m4a,3gp,3g2,mj2"
  }
}
```


Then we added the endpoint that generates a BBB container and makes different actions with it. It starts by cutting the video to 20 seconds, extracting audio tracks in AAC, MP3, and AC3 formats and finally combining them all into a single MP4 file.

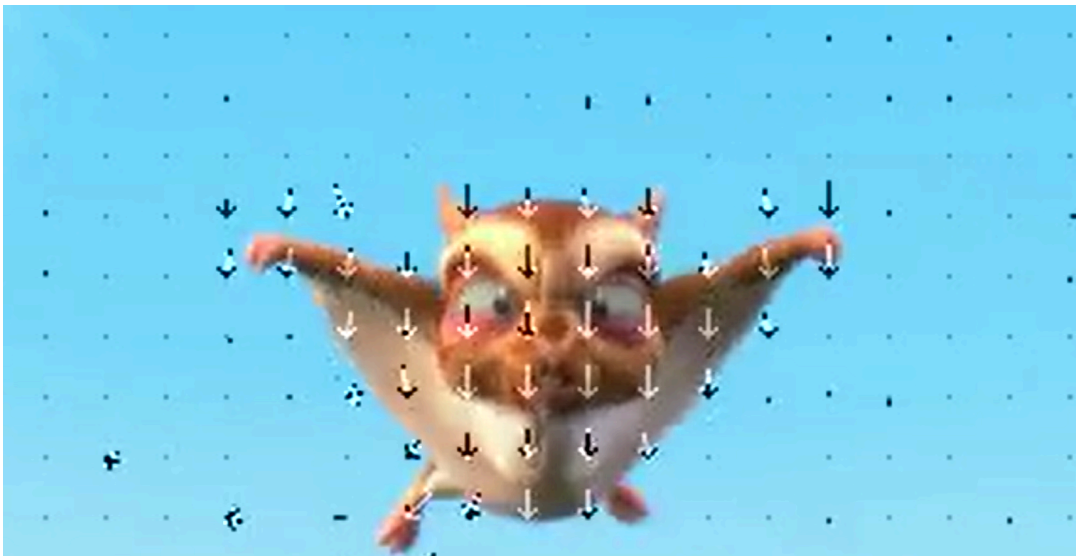

```
{
  "message": "Video processed successfully!",
  "download_links": {
    "cut_video": "http://127.0.0.1:8000/Download?file_path=processed/container_de_javi_20s.mp4",
    "mp3_audio": "http://127.0.0.1:8000/Download?file_path=processed/container_de_javi_audio.mp3",
    "aac_audio": "http://127.0.0.1:8000/Download?file_path=processed/container_de_javi_audio.aac",
    "ac3_audio": "http://127.0.0.1:8000/Download?file_path=processed/container_de_javi_audio.ac3",
    "final_video": "http://127.0.0.1:8000/Download?file_path=processed/container_de_javi_final.mp4"
  }
```

container_de_javi_final	05/12/2024 17:44	Archivo MP4	1.212 KB
container_de_javi_audio	05/12/2024 17:45	Archivo AAC	178 KB
container_de_javi_audio	05/12/2024 17:47	Archivo AC3	470 KB
container_de_javi_audio	05/12/2024 17:47	Archivo MP3	314 KB
container_de_javi_20s	05/12/2024 17:47	Archivo MP4	756 KB

The feature number five, as well as the second one, does not deliver any output file. But insead, it reads the tracks from an MP4 container and returns the number of tracks in the container.


```
{
  "message": "Track info extracted",
  "track_info": {
    "video_tracks": 1,
    "audio_tracks": 1
  }
}
```


The sixth does return a video and this one shows the macroblocks and the motion vectors of the input video.



We have also tried the feature with other videos because I found interesting how the motion vectors worked. Here is a frame of the Tintin movie:

Finally, the last feature generates a YUV histogram visualization of the input video. It returns a download link to access the processed video.