

That's correct. The code chunk you provided does not contain any information or elements directly related to the video you're looking for. The provided code is mainly focused on analytics, performance improvements, and some advertising functionality. To help you better, please provide any additional information or code related to the video in question.



node js code to download videos from a youtube playlist. You'll need my youtube cookies, which I'll provide as a Netscape HTTP Cookie File in the following format:

```
.youtube.com TRUE / FALSE 1695046429 APISID
J5UblZYY2kJ6cUX9/AqLN8FmeXe8-xJMCw
.youtube.com TRUE / TRUE 1681938458 CONSISTENCY
```

To download videos from a YouTube playlist using Node.js, you can use the `'youtube-dl'` library and `'fs'` module for saving the files. Make sure to have the `'youtube-dl'` library installed globally or as a dependency in your project. You can install it via npm:

css

Copy code

```
npm install youtube-dl
```

Please note that downloading videos from YouTube may violate their terms of service. Use the following code for educational purposes only:

javascript

Copy code

```
const fs = require('fs');
const youtubedl = require('youtube-dl');

const playlistUrl = 'https://www.youtube.com/playlist';
const outputFolder = 'downloads';
const cookiesFile = 'cookies.txt'; // Your Netscape H

if (!fs.existsSync(outputFolder)) {
  fs.mkdirSync(outputFolder);
}

youtubedl.exec(
  playlistUrl,
  [
    '-f', 'best',
    '--cookies', cookiesFile,
    '--get-id',
  ],
  {},
  (err, videoIds) => {
    if (err) throw err;

    videoIds.forEach((videoId, index) => {
      const videoUrl = `https://www.youtube.com/watch

      const video = youtubedl(videoUrl, [
        '-f', 'best',
        '--cookies', cookiesFile,
      ]);

      video.on('info', (info) => {
        console.log(`Downloading: ${info.title}`);
        video.pipe(fs.createWriteStream(`${outputFold

      video.on('complete', (info) => {
        console.log(`Finished downloading: ${info.tit

      video.on('error', (error) => {
        console.log(`Error downloading: ${error}`);
      });
    });
  });
```

```

    if (index === videoIds.length - 1) {
      video.on('end', () => {
        console.log('All videos have been downloade
      });
    }
  });
}
);

```

Replace ``YOUR_PLAYLIST_ID`` with the actual playlist ID you want to download. This script will download the videos from the playlist and save them in the ``downloads`` folder in the best available format.



How do i change just this line of code to make it accept the playlist URL as an argument when I call node download.js (where I put the code you gave me):

`https://www.youtube.com/playlist?list=PL22J3VaeABQD_Izs7y60I3IUrrFTzkpat`

You can use the ``process.argv`` property to access command line arguments in your Node.js script. Replace the ``playlistUrl`` line with the following code to accept the playlist URL as an argument:



javascript Copy code

```
const playlistUrl = process.argv[2];
```

Now, when you call your script using ``node download.js``, you can pass the playlist URL as an argument:

arduino Copy code

```
node download.js https://www.youtube.com/playlist?lis
```

Here's the modified code:

javascript Copy code

```

const fs = require('fs');
const youtubedl = require('youtube-dl');

const playlistUrl = process.argv[2];
const outputFolder = 'downloads';
const cookiesFile = 'cookies.txt'; // Your Netscape H

if (!fs.existsSync(outputFolder)) {
  fs.mkdirSync(outputFolder);
}

youtubedl.exec(
  playlistUrl,
  [
    '-f', 'best',
    '--cookies', cookiesFile,
    '--get-id',
  ],
  {},
  (err, videoIds) => {
    if (err) throw err;

    videoIds.forEach((videoId, index) => {
      const videoUrl = `https://www.youtube

```



I just get the following when I try and run it:

```

zsh: no matches found: https://www.youtube.com/playlist?list=PL22J3VaeABQD_Izs7y60I3IUrrFTzkpat

```

---

This issue is caused by the shell (zsh in this case) trying to interpret special characters, such as `?` and `&`, in the URL. To resolve this, you need to wrap the URL in single quotes when passing it as an argument:



sh

Copy code

```
node download.js 'https://www.youtube.com/playlist?li
```

This will prevent the shell from interpreting the special characters, and your script should run as expected.



Error: Command failed with exit code 1:  
/Users/cmcculloh/Projects/video-maker/node\_modules/youtube-dl/bin/youtube-dl -i -f best --cookies cookies.txt --get-id https://www.youtube.com/playlist?list=PL22J3VaeABQD\_Izs7y6OI3lUrrFTzkpat  
ERROR: Unable to extract uploader id; please report this issue on <https://yt-dl.org/bug>. Make sure you are using the latest version; type `youtube-dl -U` to update. Be sure to call youtube-dl with the `--verbose` flag and include its complete output.

---

It seems that the error is related to the `youtube-dl` library. This error might be caused by an outdated version of the library or a recent change in YouTube's HTML structure, which the library is unable to parse.



First, try updating the `youtube-dl` library. You can do this by running the following command:

sh

Copy code

```
npm install youtube-dl@latest
```

If updating the library doesn't resolve the issue, you might need to use a different library to download videos from a YouTube playlist.

One alternative is to use the `yt-dlp` library, which is an actively maintained fork of `youtube-dl`. You can install `yt-dlp` by running:

sh

Copy code

```
npm install yt-dlp
```

Then, modify the `download.js` script to use `yt-dlp` instead of `youtube-dl`. Replace the following line:

javascript

Copy code

```
const youtubedl = require('youtube-dl');
```

with:

javascript

Copy code

```
const youtubedl = require('yt-dlp');
```

This should resolve the issue, and you should be able to download videos from the YouTube playlist without encountering the error.

---



```
/Users/cmcculloh/Projects/video-maker/playlist_download.cjs:2
const youtubedl = require("yt-dlp");
^
```

```
Error [ERR_REQUIRE_ESM]: require() of ES Module
/Users/cmcculloh/Projects/video-maker/node_modules/yt-
dlp/app/index.js from /Users/cmcculloh/Projects/video-
maker/playlist_download.cjs not supported.
Instead change the require of index.js in
/Users/cmcculloh/Projects/video-maker/playlist_download.cjs to
a dynamic import() which is available in all CommonJS modules.
    at Object.<anonymous> (/Users/cmcculloh/Projects/video-
maker/playlist_download.cjs:2:19) {
  code: 'ERR_REQUIRE_ESM'
}
```

Apologies for the confusion. The `yt-dlp` library is an ES module, and it requires a different way to import it in a CommonJS module. You can use the `import()` function to load the library.



First, change the file extension of your `playlist_download` script from `.cjs` to `.mjs` (for example, `playlist_download.mjs`). This will allow you to use ES module syntax.

Next, update the `playlist_download.mjs` script to use dynamic `import()` instead of `require()` for importing `yt-dlp`. Replace the following line:

```
javascript  Copy code
```

```
const youtubedl = require('yt-dlp');
```

with:

```
javascript  Copy code
```

```
import youtubedl from 'yt-dlp';
```

Finally, run the script using Node.js with the following command:

```
sh  Copy code
```

```
node playlist_download.mjs
```

This should resolve the issue, and you should be able to use the `yt-dlp` library without encountering the error.



```
file:///Users/cmcculloh/Projects/video-
maker/playlist_download.mjs:2
import youtubedl from "yt-dlp";
^^^^^^^^
```

```
SyntaxError: The requested module 'yt-dlp' does not provide an
export named 'default'
    at ModuleJob.instantiate
(node:internal/modules/esm/module_job:127:21)
    at async ModuleJob.run
(node:internal/modules/esm/module_job:193:5)
    at async Promise.all (index 0)
    at async ESMLoader.import
(node:internal/modules/esm/loader:337:24)
    at async loadESM (node:internal/process/esm_loader:88:5)
    at async handleMainPromise
(node:internal/modules/run_main:61:12)
```

Apologies for the confusion earlier. Since `yt-dlp` does not have a default export, we need to change the import syntax.

Please modify the `playlist_download.mjs` script as follows:

javascript

 Copy code

```
import * as youtubedl from 'yt-dlp';

// The rest of the code remains the same
```

This should import the `'yt-dlp'` library correctly. Then, you can run the script using the following command:

sh

 Copy code

```
node playlist_download.mjs
```

This should resolve the issue and allow you to use the `'yt-dlp'` library without encountering any errors.



```
file:///Users/cmcculloh/Projects/video-
maker/playlist_download.mjs:12
youtubedl.exec(
  ^
```

```
TypeError: youtubedl.exec is not a function
    at file:///Users/cmcculloh/Projects/video-
maker/playlist_download.mjs:12:11
    at ModuleJob.run
(node:internal/modules/esm/module_job:197:25)
    at async Promise.all (index 0)
    at async ESMLoader.import
(node:internal/modules/esm/loader:337:24)
    at async loadESM (node:internal/process/esm_loader:88:5)
    at async handleMainPromise
(node:internal/modules/run_main:61:12)
```

---

I apologize for the confusion. It seems that ``yt-dlp`` does not have an ``exec`` method when using the ESM import. Instead, we can use the ``child_process`` module to run the ``yt-dlp`` command. Please modify the ``playlist_download.mjs`` script as follows:



javascript Copy code

```
import { exec } from 'child_process';
import util from 'util';
const execPromisified = util.promisify(exec);

const playlistUrl = process.argv[2];
const cookiesFilePath = 'cookies.txt';

(async () => {
  try {
    const { stdout: ids } = await execPromisified(`yt

    const videoIds = ids.split('\n').filter(id => id)

    for (const videoId of videoIds) {
      const { stdout: videoUrl } = await execPromisif
      console.log(`Video URL: ${videoUrl}`);
    }
  } catch (error) {
    console.error(`Error: ${error}`);
  }
})();
```

Then, you can run the script using the following command:

sh Copy code

```
node playlist_download.mjs "https://www.youtube.com/p
```

This should resolve the issue and correctly display the video URLs for each video in the playlist.



It didn't download them though, how do I make it download them?

To download the videos, you can use the ``fs`` module to write the content of the videos to local files. Modify the ``playlist_download.mjs`` script as follows:

javascript Copy code

```
import { exec } from 'child_process';
import util from 'util';
import fs from 'fs';
import https from 'https';
const execPromisified = util.promisify(exec);
const writeFile = util.promisify(fs.writeFile);

const playlistUrl = process.argv[2];
const cookiesFilePath = 'cookies.txt';

async function downloadVideo(url, filename) {
  return new Promise((resolve, reject) => {
    const file = fs.createWriteStream(filename);
    https.get(url, (response) => {
      response.pipe(file);
      file.on('finish', () => {
        file.close(resolve);
      });
    }).on('error', (error) => {
      fs.unlink(filename);
      reject(error);
    });
  });
}
```

```
(async () => {
  try {
    const { stdout: ids } = await execPromisified(`yt

    const videoIds = ids.split('\n').filter(id => id)

    for (const videoId of videoIds) {
      const { stdout: videoUrl } = await execPromisified
      console.log(`Downloading video: ${videoId}`);
      await downloadVideo(videoUrl.trim(), `${videoId}
      console.log(`Downloaded video: ${videoId}`);
    }
  } catch (error) {
    console.error(`Error: ${error}`);
  }
})();
```

This script will now download the videos to the same directory as the script, with each video file named after its video ID and saved in the MP4 format. Run the script using the following command:

```
sh
node playlist_download.mjs "https://www.youtube.com/p
```

The script will download each video and display a message when the download is complete.



I want it to place them in a "downloads" directory, show me just the change I would have to make to do that

To save the downloaded videos in a "downloads" directory, you only need to modify the `downloadVideo` function call in the loop. Here's the change you need to make:



Change this line:

```
javascript
await downloadVideo(videoUrl.trim(), `${videoId}.mp4`
```

To this:

```
javascript
await downloadVideo(videoUrl.trim(), `downloads/${vid
```

Before running the script, make sure to create a "downloads" directory in the same folder as your script, or add the following lines at the beginning of your script to create the directory programmatically:

```
javascript
import fs from 'fs';

if (!fs.existsSync('downloads')) {
  fs.mkdirSync('downloads');
}
```

Now, when you run the script, it will save the downloaded videos in the "downloads" directory.



Can I make the video be named after the title of the video on youtube?

Yes, you can do that. You'll need to modify the `getVideoIds` function to return video titles along with their IDs. Then, use the title for naming the downloaded videos. Here's the updated code:



1. Modify the `getVideoIds` function to include video titles in the returned array:

javascript Copy code

```
async function getVideoIds(playlistUrl) {
  const output = await youtubedl(playlistUrl, {
    dumpSingleJson: true,
    flatPlaylist: true,
    noWarnings: true,
    noCallHome: true,
    noCheckCertificate: true,
    preferFreeFormats: true,
    youtubeSkipDashManifest: true,
  });

  const videos = output.entries.map((entry) => ({
    id: entry.id,
    title: entry.title,
  }));

  return videos;
}
```

2. Update the loop to use the video title for naming the downloaded files:

javascript Copy code

```
for (const { id, title } of videoIds) {
  const videoUrl = `https://www.youtube.com/watch?v=${id}`;
  const fileName = `${title.replace(/[\/\\?%*:|'<>]/g, '')}.mp4`;
  await downloadVideo(videoUrl.trim(), `downloads/${fileName}`);
}
```

The `replace` function in the `fileName` line is used to remove any characters that are not allowed in file names. The downloaded videos will now be saved in the "downloads" directory with their respective titles as file names.



Is there a way to modify this code to make the fileName be the video title instead of id?

```
import { exec } from "child_process";
import util from "util";
import fs from "fs";
import https from "https";
const execPromisified = util.promisify(exec);
const writeFile = util.promisify(fs.writeFile);

const playlistUrl = process.argv[2];
const cookiesFilePath = "cookies.txt";

if (!fs.existsSync("downloads")) {
  fs.mkdirSync("downloads");
}

async function downloadVideo(url, filename) {
  return new Promise((resolve, reject) => {
    const file = fs.createWriteStream(filename);
    https
      .get(url, (response) => {
        response.pipe(file);
        file.on("finish", () => {
          file.close(resolve);
        });
      })
      .on("error", (error) => {
        fs.unlink(filename);
        reject(error);
      });
  });
}
```



```

    });
  }

  (async () => {
    try {
      const { stdout: ids } = await execPromisified(
        `yt-dlp -i -f best --cookies ${cookiesFilePath} --get-id ${playlistUrl}`
      );

      const videoIds = ids.split("\n").filter((id) => id);

      for (const videoId of videoIds) {
        const { stdout: videoUrl } = await execPromisified(
          `yt-dlp -i -f best --cookies ${cookiesFilePath} --get-url https://www.youtube.com/watch?v=${videoId}`
        );
        console.log(`Downloading video: ${videoId}`);
        await downloadVideo(videoUrl.trim(),
          `downloads/${videoId}.mp4`);
        console.log(`Downloaded video: ${videoId}`);
      }
    } catch (error) {
      console.error(`Error: ${error}`);
    }
  })();
}

```

Yes, you can modify the code to save the video with its title instead of the video ID. You'll need to first fetch the video title using `yt-dlp`. Here's the updated code:



1. Add a function to get the video title:

```

javascript Copy code

async function getVideoTitle(videoId) {
  const { stdout: title } = await execPromisified(
    `yt-dlp --cookies ${cookiesFilePath} --get-title`
  );
  return title.trim();
}

```

2. Update the loop to use the video title for naming the downloaded files:

```

javascript Copy code

for (const videoId of videoIds) {
  const { stdout: videoUrl } = await execPromisified(
    `yt-dlp -i -f best --cookies ${cookiesFilePath} -`
  );

  const title = await getVideoTitle(videoId);
  const fileName = `${title.replace(/[\/\?%*:|"<>]/g,
    console.log(`Downloading video: ${title}`);
    await downloadVideo(videoUrl.trim(), `downloads/${f
    console.log(`Downloaded video: ${title}`);
  }
}

```

The `replace` function in the `fileName` line is used to remove any characters that are not allowed in file names. The downloaded videos will now be saved in the "downloads" directory with their respective titles as file names.



OK, that's good, but I want to modify it further. This time I want each filename to have an episode number pre-pended to it so that it will display properly in my Plex server, with the episodes numbered in chronological order. Determine which season the episode is part of by what year the video was published and assume the oldest video begins season 1

To achieve this, you'll need to fetch the video upload date and calculate the season based on the year it was published. Here's the modified code:



1. Add a function to get the video upload date:

```
javascript Copy code

async function getVideoUploadDate(videoId) {
  const { stdout: uploadDate } = await execPromisified(
    `yt-dlp --cookies ${cookiesFilePath} --get-upload`
  );
  return uploadDate.trim();
}
```

2. Add a helper function to calculate the season:

```
javascript Copy code

function getSeason(uploadDate) {
  const uploadYear = parseInt(uploadDate.slice(0, 4),
  const startYear = 2005; // Replace with the actual
  return uploadYear - startYear + 1;
}
```

3. Update the loop to use episode numbers and seasons:

```
javascript Copy code

let episodeCounts = {};

for (const videoId of videoIds) {
  const { stdout: videoUrl } = await execPromisified(
    `yt-dlp -i -f best --cookies ${cookiesFilePath} -`
  );

  const title = await getVideoTitle(videoId);
  const uploadDate = await getVideoUploadDate(videoId);
  const season = getSeason(uploadDate);

  if (!episodeCounts[season]) {
    episodeCounts[season] = 1;
  } else {
    episodeCounts[season]++;
  }

  const episodeNumber = episodeCounts[season];
  const fileName = `${season.toString().padStart(2,
  console.log(`Downloading video: ${title}`);
  await downloadVideo(videoUrl.trim(), `downloads/${f
  console.log(`Downloaded video: ${title}`);
}
```

Replace `'2005'` with the actual starting year of your video series in the `getSeason` function.

Now, the downloaded videos will have filenames with the format `'SXXEXX - Title.mp4'`, where `'XX'` are the season and episode numbers, respectively. The episodes will be numbered based on the chronological order of their upload dates.



When I tried to run that, I got this:

```
Error: Error: Command failed: yt-dlp --cookies cookies.txt --get-
upload-date https://www.youtube.com/watch?v=j1lo59UG3y0
```

```
Usage: yt-dlp [OPTIONS] URL [URL...]
```

```
yt-dlp: error: no such option: --get-upload-date
```

Apologies for the confusion. It seems that `--get-upload-date` is not a valid option for yt-dlp. We can use the JSON output and extract the necessary information. Here's the updated code:

1. Replace the `getVideoUploadDate` function with the following:

```
javascript Copy code  
  
async function getVideoInfo(videoId) {  
  const { stdout: jsonOutput } = await execPromisified(  
    `yt-dlp --cookies ${cookiesFilePath} -j https://w  
  );  
  return JSON.parse(jsonOutput);  
}
```

This new function will fetch the entire video information as a JSON object.

2. Update the loop to use the new `getVideoInfo` function and extract the upload date:

```
javascript Copy code  
  
let episodeCounts = {};  
  
for (const videoId of videoIds) {  
  const { stdout: videoUrl } = await execPromisified(  
    `yt-dlp -i -f best --cookies ${cookiesFilePath} -  
  );  
  
  const videoInfo = await getVideoInfo(videoId);  
  const title = videoInfo.title;  
  const uploadDate = videoInfo.upload_date;  
  const season = getSeason(uploadDate);  
  
  if (!episodeCounts[season]) {  
    episodeCounts[season] = 1;  
  } else {  
    episodeCounts[season]++;  
  }  
  
  const episodeNumber = episodeCounts[season];  
  const fileName = `${season.toString().padStart(2,  
  
  console.log(`Downloading video: ${title}`);  
  await downloadVideo(videoUrl.trim(), `downloads/${f  
  console.log(`Downloaded video: ${title}`);  
}
```

Now, the code should work as expected. The `getVideoInfo` function fetches the video information, and the loop extracts the title and upload date from the JSON object.



It seems like everything downloaded OK, but I got this at the end:

Error: RangeError [ERR\_CHILD\_PROCESS\_STDIO\_MAXBUFFER]:  
stdout maxBuffer length exceeded

This error occurs when the output from a child process exceeds the default `maxBuffer` size (which is 1024 \* 1024 bytes). To resolve this issue, we can increase the `maxBuffer` size for the `execPromisified` function.

Here's how to modify the `execPromisified` calls to increase the `maxBuffer` size:

1. Replace the following line:

```
javascript Copy code
```

```
const { stdout: ids } = await execPromisified(`yt-dlp
```

with:

```
javascript  Copy code
```

```
const { stdout: ids } = await execPromisified(`yt-dlp
```

2. Replace the following line:

```
javascript  Copy code
```

```
const { stdout: videoUrl } = await execPromisified(`y
```

with:

```
javascript  Copy code
```

```
const { stdout: videoUrl } = await execPromisified(`y
```

3. Replace the following line:

```
javascript  Copy code
```

```
const { stdout: jsonOutput } = await execPromisified(
```

with:

```
javascript  Copy code
```

```
const { stdout: jsonOutput } = await execPromisified(
```

These modifications increase the `maxBuffer` size to 10MB, which should be sufficient for most use cases. If you still encounter the error, you can increase the `maxBuffer` size further.



ok, this is all great. However, it is labeling the videos in reverse-chronological order. So, S01E08 should really be S01E01, and S01E01 should be S01E08. I think we need to change the way we get the episodes or calculate them. Maybe process them in reverse order?

You can reverse the order of the `videoids` array before processing them. This way, the oldest video will be labeled as S01E01, and so on. To do this, just add the following line after filtering the `videoids`:

```
javascript  Copy code
```

```
videoIds.reverse();
```

Here's the modified code snippet:

```
javascript  Copy code
```

```
const videoIds = ids.split("\n").filter((id) => id);
```

```
// Reverse the order of videoIds  
videoIds.reverse();
```

```
for (const videoId of videoIds) {  
  // The rest of the code remains the same  
}
```

This modification should ensure that the videos are labeled in chronological order, with the oldest video starting as S01E01.



Perfect, thanks! ok, now can we capture and save to file the ids of the downloaded videos, and then next time the playlist is requested make sure to only download \_new\_ videos?

Sure! You can save the video IDs of the downloaded videos to a file, and then read the file to check for new videos when running the script again. Here's the modified code:

```
javascript Copy code

import { exec } from "child_process";
import util from "util";
import fs from "fs";
import https from "https";
const execPromisified = util.promisify(exec);
const writeFile = util.promisify(fs.writeFile);
const readFile = util.promisify(fs.readFile);

const playlistUrl = process.argv[2];
const cookiesFilePath = "cookies.txt";
const downloadedVideosFile = "downloaded_videos.txt";

if (!fs.existsSync("downloads")) {
  fs.mkdirSync("downloads");
}

async function downloadVideo(url, filename) {
  // ... downloadVideo function remains the same
}

(async () => {
  try {
    const { stdout: ids } = await execPromisified(
      `yt-dlp -i -f best --cookies ${cookiesFil
    );

    let downloadedIds = [];
    if (fs.existsSync(downloadedVideosFile)) {
      const content = await readFile(downloaded
      downloadedIds = content.split("\n").filte
    }

    const videoIds = ids
      .split("\n")
      .filter((id) => id && !downloadedIds.incl
      .reverse();

    for (const videoId of videoIds) {
      // ... Downloading videos code remains th

      // Append downloaded videoId to the file
      await writeFile(downloadedVideosFile, `${
    }
  } catch (error) {
    console.error(`Error: ${error}`);
  }
})();
```

This code adds a new file called `downloaded_videos.txt` to store the video IDs of the downloaded videos. When running the script, it reads the file to get the list of downloaded videos and filters out the already downloaded ones from the playlist. Finally, it appends the newly downloaded video ID to the file.

Now, when you run the script again, it will only download the new videos from the playlist.



Can we make it so that the `downloaded_videos.txt` file gets saved into the `downloads` folder and gets named after the passed in URL somehow?

Sure, we can save the `'downloaded_videos.txt'` file in the `'downloads'` folder and name it based on the passed-in playlist URL. Here's the modified code:



javascript

Copy code

```
import { exec } from "child_process";
import util from "util";
import fs from "fs";
import https from "https";
const execPromisified = util.promisify(exec);
const writeFile = util.promisify(fs.writeFile);
const readFile = util.promisify(fs.readFile);

const playlistUrl = process.argv[2];
const cookiesFilePath = "cookies.txt";

if (!fs.existsSync("downloads")) {
  fs.mkdirSync("downloads");
}

// Generate a unique filename based on the playlist URL
const playlistId = new URL(playlistUrl).searchParams.get("list");
const downloadedVideosFile = `downloads/downloaded_videos_${playlistId}.txt`;

async function downloadVideo(url, filename) {
  // ... downloadVideo function remains the same
}

(async () => {
  try {
    const { stdout: ids } = await execPromisified(
      `yt-dlp -i -f best --cookies ${cookiesFilePath} ${playlistUrl}`
    );

    let downloadedIds = [];
    if (fs.existsSync(downloadedVideosFile)) {
      const content = await readFile(downloadedVideosFile);
      downloadedIds = content.split("\n").filter(id => id);
    }

    const videoIds = ids
      .split("\n")
      .filter(id => id && !downloadedIds.includes(id))
      .reverse();

    for (const videoId of videoIds) {
      // ... Downloading videos code remains the same

      // Append downloaded videoId to the file
      await writeFile(downloadedVideosFile, `${videoId}\n`, {
        append: true,
      });
    }
  } catch (error) {
    console.error(`Error: ${error}`);
  }
})();
```

Now, the `'downloaded_videos.txt'` file will be saved in the `'downloads'` folder and named after the playlist ID extracted from the passed-in URL.



Will the new video downloads be named properly, with the correct Season and Episode number? Or do we need to somehow cache or track that data in the `downloaded_videos.txt` file? We might even need to make that file a JSON file and change the structure slightly, but I'll leave that up to you.

You're right; we need to store the season and episode information for the videos so that when new videos are

downloaded, they have the correct season and episode numbers. We can change the `downloaded_videos.txt` file to a JSON file to store this information. Here's the modified code:

```
javascript Copy code

import { exec } from "child_process";
import util from "util";
import fs from "fs";
import https from "https";
const execPromisified = util.promisify(exec);
const writeFile = util.promisify(fs.writeFile);
const readFile = util.promisify(fs.readFile);

const playlistUrl = process.argv[2];
const cookiesFilePath = "cookies.txt";

if (!fs.existsSync("downloads")) {
  fs.mkdirSync("downloads");
}

// Generate a unique filename based on the playlist URL
const playlistId = new URL(playlistUrl).searchParams.get("v");
const downloadedVideosFile = `downloads/downloaded_videos_${playlistId}.json`;

async function downloadVideo(url, filename) {
  // ... downloadVideo function remains the same
}

(async () => {
  try {
    const { stdout: ids } = await execPromisified(
      `yt-dlp -i -f best --cookies ${cookiesFilePath} ${playlistUrl}`
    );

    let downloadedData = [];
    if (fs.existsSync(downloadedVideosFile)) {
      const content = await readFile(downloadedVideosFile);
      downloadedData = JSON.parse(content);
    }

    const videoIds = ids
      .split("\n")
      .filter((id) => id && !downloadedData.some((video) => video.id === id))
      .reverse();

    let seasonCounter = downloadedData.length > 0 ? downloadedData[downloadedData.length - 1].season : 0;
    let episodeCounter = downloadedData.length > 0 ? downloadedData[downloadedData.length - 1].episode : 0;

    for (const videoId of videoIds) {
      // ... Downloading videos code remains the same

      const fileName = `downloads/${videoId}.mp4`;

      // ... Rest of the downloading videos code

      // Append downloaded video data to the file
      downloadedData.push({ id: videoId, season: seasonCounter, episode: episodeCounter });
      await writeFile(downloadedVideosFile, JSON.stringify(downloadedData, null, 2));
      episodeCounter++;
    }
  } catch (error) {
    console.error(`Error: ${error}`);
  }
})();
```

Now, the `downloaded_videos.json` file will store the video ID, season, and episode number for each downloaded video. When new videos are downloaded, the code will use this information to calculate the correct season and episode numbers.



FYI, I changed the `downloadedData.push` line to look like this:

```
// Append downloaded video data to the file
downloadedData.push({
  id: videoId,
```

```
season: seasonCounter,
episode: episodeCounter,
url: videoUrl.trim(),
});
```

I just wanted to let you know for next time we use this code. You don't have to say anything other than "understood" to confirm that you acknowledge.

Understood. Thanks for the update! If you have any further questions or need more assistance, feel free to ask.



This code:

```
// Generate a unique filename based on the playlist URL
const playlistId = new URL(playlistUrl).searchParams.get("list");
const downloadedVideosFile =
`downloads/downloaded_videos_${playlistId}.json`;
```

Should also be able to accommodate video listing pages like the following URL:  
<https://www.youtube.com/@TrueNorthOutdoorAdventures/videos>

I understand your concern. You can use the following modification to handle both playlist URLs and video listing pages like the one you mentioned:



javascript

Copy code

```
let downloadedVideosFile;

if (playlistUrl.includes("playlist")) {
  const playlistId = new URL(playlistUrl).searchParam
  downloadedVideosFile = `downloads/downloaded_videos
} else if (playlistUrl.includes("videos")) {
  const channelId = playlistUrl.split("/").pop();
  downloadedVideosFile = `downloads/downloaded_videos
} else {
  console.error("Invalid URL format. Please provide a
process.exit(1);
}
```

This code checks if the given URL is a playlist or a video listing page and generates an appropriate filename accordingly. If the URL doesn't match either of these formats, it will display an error message and exit the process.



"channelId" ends up just being "videos" for some reason. I'd rather it be the actual name of the channel. You don't have to give me all of the new code just give me the updates I need to make.

I apologize for the confusion. In that case, we'll need to get the channel name first. You can use the following command to obtain the channel name using `yt-dlp`:

bash

Copy code

```
yt-dlp --cookies cookies.txt --get-filename --no-play
```

Replace `<URL>` with the actual URL. Now, you can integrate this command into your code to get the channel name and use it as the channelId:

javascript

Copy code

```
// Replace the previous channelId assignment with the
let channelId;
if (playlistUrl.includes("videos")) {
```



Now, the `channelId` variable should contain the actual channel name. Remember to add this code within the same async block where other `execPromisified` calls are being made.



running that ends up with:

[illegible]

I apologize for the confusion. It seems like the output from the command is returning multiple lines. We need to pick the first line as the channel name. You can update the code as follows:

```

// Replace the previous channelId assignment with the
let channelId;

if (playlistUrl.includes("videos")) {
  try {
    const { stdout: channelName } = await execPromise(
      `yt-dlp --cookies ${cookiesFilePath} --get-file`
    );
    channelId = channelName.trim().split("\n")[0];
  } catch (error) {
    console.error("Error getting channel name: ${error}");
    process.exit(1);
  }
}

```

```
}  
}
```

This should assign the correct channel name to the `channelId` variable.



Perfect! But they are all saying they are in S01 instead of being in different seasons.

I apologize for the confusion. It looks like I made a mistake in the previous code. To fix the issue, update the season calculation logic in the for loop as follows:



Replace this line:

```
javascript  Copy code  
  
const seasonCounter = new Date(uploadDate).getFullYear()
```

With:

```
javascript  Copy code  
  
const seasonCounter = new Date(uploadDate).getFullYear()
```

This should correctly calculate the season number based on the difference in years between the upload dates of the videos.



I think you've lost track of how the code looks at this point. Here's the code, please suggest the changes that need to be made to this code to get the season number to be correct:

```
import { exec } from "child_process";  
import util from "util";  
import fs from "fs";  
import https from "https";  
const execPromisified = util.promisify(exec);  
const writeFile = util.promisify(fs.writeFile);  
const readFile = util.promisify(fs.readFile);  
  
const playlistUrl = process.argv[2];  
const cookiesFilePath = "cookies.txt";  
  
if (!fs.existsSync("downloads")) {  
  fs.mkdirSync("downloads");  
}  
  
// Generate a unique filename based on the playlist URL  
let downloadedVideosFile;  
let outputDir;  
  
if (playlistUrl.includes("playlist")) {  
  const playlistId = new URL(playlistUrl).searchParams.get("list");  
  downloadedVideosFile =  
    `downloads/downloaded_videos_${playlistId}.json`;  
  outputDir = `downloads/${playlistId}`;  
} else if (playlistUrl.includes("videos")) {  
  let channelName;  
  
  try {  
    ({ stdout: channelName } = await execPromisified(  
      `yt-dlp --cookies ${cookiesFilePath} --get-filename --no-  
playlist --output "%(uploader)s" "${playlistUrl}"`  
    ));  
    channelName = channelName.trim().split("\n")[0];  
    console.log("channelName", channelName);  
  } catch (error) {  
    console.error(`Error getting channel name: ${error}`);  
    process.exit(1);  
  }  
}
```

```

downloadedVideosFile =
`downloads/downloaded_videos_${channelName}.json`;
outputDir = `downloads/${channelName}`;
} else {
  console.error("Invalid URL format. Please provide a playlist or
videos URL.");
  process.exit(1);
}

if (fs.existsSync(outputDir)) {
  fs.mkdirSync(outputDir);
}

async function downloadVideo(url, filename) {
  return new Promise((resolve, reject) => {
    const file = fs.createWriteStream(filename);
    https
      .get(url, (response) => {
        response.pipe(file);
        file.on("finish", () => {
          file.close(resolve);
        });
      })
      .on("error", (error) => {
        fs.unlink(filename);
        reject(error);
      });
  });
}

async function getVideoInfo(videoId) {
  const { stdout: jsonOutput } = await execPromisified(
    `yt-dlp --cookies ${cookiesFilePath} -j
https://www.youtube.com/watch?v=${videoId}`,
    { maxBuffer: 10 * 1024 * 1024 }
  );

  return JSON.parse(jsonOutput);
}

(async () => {
  try {
    const { stdout: ids } = await execPromisified(
      `yt-dlp -i -f best --cookies ${cookiesFilePath} --get-id
${playlistUrl}`
    );

    let downloadedData = [];
    if (fs.existsSync(downloadedVideosFile)) {
      const content = await readFile(downloadedVideosFile, "utf-
8");
      downloadedData = JSON.parse(content);
    }

    const videoIds = ids
      .split("\n")
      .filter((id) => id && !downloadedData.some((entry) =>
entry.id === id))
      .reverse();

    let seasonCounter =
      downloadedData.length > 0 ?
downloadedData[downloadedData.length - 1].season : 1;
    let episodeCounter =
      downloadedData.length > 0 ?
downloadedData[downloadedData.length - 1].episode + 1 : 1;

    for (const videoId of videoIds) {
      const { stdout: videoUrl } = await execPromisified(
        `yt-dlp -i -f best --cookies ${cookiesFilePath} --get-url
https://www.youtube.com/watch?v=${videoId}`,
        { maxBuffer: 10 * 1024 * 1024 }
      );

      const videoInfo = await getVideoInfo(videoId);
      const title = videoInfo.title;

      const fileName = `${outputDir}/${S}${seasonCounter
.toString()
.padStart(2,
"0"))E${episodeCounter.toString().padStart(2,

```

```

"0"))-${title.replace(
    /\[\/.*?<>[]/g,
    " "
  )}.mp4`;

  console.log(` Downloading video: ${title}`);
  await downloadVideo(videoUrl.trim(), fileName);
  console.log(` Downloaded video: ${title}`);

  // Append downloaded video data to the file
  downloadedData.push({
    id: videoId,
    season: seasonCounter,
    episode: episodeCounter,
    url: videoUrl.trim(),
  });
  await writeFile(downloadedVideosFile,
    JSON.stringify(downloadedData, null, 2));


  episodeCounter++;
}
} catch (error) {
  console.error(` Error: ${error}`);
}
})();

```

I apologize for the confusion. To properly calculate the season number, you need to first get the oldest video's upload date before the main loop and then use it to calculate the season number for each video in the loop. Please make the following changes to the code:

1. Before the main loop, add this code to get the oldest video's upload date:

```


javascript  Copy code

const firstVideoId = videoIds[videoIds.length - 1];
const firstVideoInfo = await getVideoInfo(firstVideoId);
const firstVideoUploadDate = firstVideoInfo.upload_date;

```

2. Inside the main loop, replace the season and episode counter initialization with this code:

```

javascript  Copy code

const videoInfo = await getVideoInfo(videoId);
const title = videoInfo.title;
const uploadDate = videoInfo.upload_date;

const seasonCounter = new Date(uploadDate).getFullYear();

```

3. Remove the `seasonCounter` and `episodeCounter` initializations before the loop since they will be calculated for each video inside the loop.

Now the code should calculate the correct season number for each video based on the difference in years between the upload dates. Here's the updated code with the changes:

```

javascript  Copy code

import { exec } from "child_process";
import util from "util";
import fs from "fs";
import https from "https";
const execPromisified = util.promisify(exec);
const writeFile = util.promisify(fs.writeFile);
const readFile = util.promisify(fs.readFile);

const playlistUrl = process.argv[2];
const cookiesFilePath = "cookies.txt";

// ... (Rest of the code before the loop)

```

```

// Get the oldest video's upload date
const firstVideoId = videoIds[videoIds.length - 1];
const firstVideoInfo = await getVideoInfo(firstVideoId);
const firstVideoUploadDate = firstVideoInfo.upload_date;

for (const videoId of videoIds) {
  const { stdout: videoUrl } = await execPromisified(
    `yt-dlp -i -f best --cookies ${cookiesFilePath} -
    { maxBuffer: 10 * 1024 * 1024 }
  );

  const videoInfo = await getVideoInfo(videoId);
  const title = videoInfo.title;
  const uploadDate = videoInfo.upload_date;

  const seasonCounter = new Date(uploadDate).getFullYear() -
    firstVideoUploadDate;
  const episodeCounter = downloadedData.filter((entry) =>
    entry.title === title).length;

  // ... (Rest of the code in the loop)
}

```

This should correctly calculate the season number for each video based on the difference in years between the upload dates.



Can you just show me all of the updated code? It looks like something isn't lining up here...

I apologize for the confusion. Here's the updated code with the correct season and episode numbers calculation:

```

javascript Copy code

import { exec } from "child_process";
import util from "util";
import fs from "fs";
import https from "https";

const execPromisified = util.promisify(exec);
const writeFile = util.promisify(fs.writeFile);
const readFile = util.promisify(fs.readFile);

const playlistUrl = process.argv[2];
const cookiesFilePath = "cookies.txt";

// ... (Rest of the code before the loop)

// Get the oldest video's upload date
const firstVideoId = videoIds[videoIds.length - 1];
const firstVideoInfo = await getVideoInfo(firstVideoId);
const firstVideoUploadDate = firstVideoInfo.upload_date;

for (const videoId of videoIds) {
  const { stdout: videoUrl } = await execPromisified(
    `yt-dlp -i -f best --cookies ${cookiesFilePath} -
    { maxBuffer: 10 * 1024 * 1024 }
  );

  const videoInfo = await getVideoInfo(videoId);
  const title = videoInfo.title;
  const uploadDate = videoInfo.upload_date;

  const seasonCounter = new Date(uploadDate).getFullYear() -
    firstVideoUploadDate;
  const episodeCounter = downloadedData.filter((entry) =>
    entry.title === title).length;

  const fileName = `${outputDir}/${seasonCounter}
    .toString()
    .padStart(2, "0")}${episodeCounter.toString().padStart(
      2, "0"
    )}.mp4`;

  console.log(`Downloading video: ${title}`);
  await downloadVideo(videoUrl.trim(), fileName);
  console.log(`Downloaded video: ${title}`);

  // Append downloaded video data to the file
  downloadedData.push({
    title: title,
    url: videoUrl,
    season: seasonCounter,
    episode: episodeCounter,
  });
}

```

```

        id: videoId,
        season: seasonCounter,
        episode: episodeCounter,
        url: videoUrl.trim(),
    });
    await writeFile(downloadedVideosFile, JSON.stringify
    }

    // ... (Rest of the code after the loop)

```

This code should correctly calculate the season and episode numbers for each video based on the difference in years between the upload dates.



ok but show me all of the code please

Apologies for the confusion. Here is the complete code with the correct season and episode numbers calculation:

```

javascript Copy code

import { exec } from "child_process";
import util from "util";
import fs from "fs";
import https from "https";
const execPromisified = util.promisify(exec);
const writeFile = util.promisify(fs.writeFile);
const readFile = util.promisify(fs.readFile);

const playlistUrl = process.argv[2];
const cookiesFilePath = "cookies.txt";

if (!fs.existsSync("downloads")) {
    fs.mkdirSync("downloads");
}

let downloadedVideosFile;
let outputDir;

if (playlistUrl.includes("playlist")) {
    const playlistId = new URL(playlistUrl).searchParams.get("id");
    downloadedVideosFile = `downloads/downloaded_videos_${playlistId}.json`;
    outputDir = `downloads/${playlistId}`;
} else if (playlistUrl.includes("videos")) {
    let channelName;

    try {
        ({ stdout: channelName } = await execPromisified(
            `yt-dlp --cookies ${cookiesFilePath} --get-id`
        ));
        channelName = channelName.trim().split("\n")[0];
        console.log("channelName", channelName);
    } catch (error) {
        console.error("Error getting channel name: ", error);
        process.exit(1);
    }

    downloadedVideosFile = `downloads/downloaded_videos_${channelName}.json`;
    outputDir = `downloads/${channelName}`;
} else {
    console.error("Invalid URL format. Please provide a valid URL.");
    process.exit(1);
}

if (!fs.existsSync(outputDir)) {
    fs.mkdirSync(outputDir);
}

async function downloadVideo(url, filename) {
    return new Promise((resolve, reject) => {
        const file = fs.createWriteStream(filename);
        https
            .get(url, (response) => {
                response.pipe(file);
                file.on("finish", () => {
                    file.close(resolve);
                });
            })
            .on("error", reject);
    });
}

```

```

    });
  });
  .on("error", (error) => {
    fs.unlink(filename);
    reject(error);
  });
});
}

async function getVideoInfo(videoId) {
  const { stdout: jsonOutput } = await execPromisified(
    `yt-dlp --cookies ${cookiesFilePath} -j https`
    { maxBuffer: 10 * 1024 * 1024 }
  );

  return JSON.parse(jsonOutput);
}

(async () => {
  try {
    const { stdout: ids } = await execPromisified(
      `yt-dlp -i -f best --cookies ${cookiesFil`
    );

    let downloadedData = [];
    if (fs.existsSync(downloadedVideosFile)) {
      const content = await readFile(downloaded
        downloadedData = JSON.parse(content);
    }

    const videoIds = ids
      .split("\n")
      .filter((id) => id && !downloadedData.some
      .reverse();

    const firstVideoId = videoIds[videoIds.length
    const firstVideoInfo = await getVideoInfo(fir
    const firstVideoUploadDate = firstVideoInfo.u

    for (const videoId of videoIds) {
      const { stdout: videoUrl } = await execPr
        `yt-dlp -i -f best --cookies ${cookie
        { maxBuffer: 10 * 102

```



you stopped generating at:

```

for (const videoId of videoIds) {
  const { stdout: videoUrl } = await execPromisified(
    `yt-dlp -i -f best --cookies ${cookiesFilePath} --get-url`
    https://www.youtube.com/watch?v=${videoId}`,
    { maxBuffer: 10 * 102

```

please continue from there.

Apologies for the truncation. Here is the complete code from that point:

javascript

 Copy code

```

for (const videoId of videoIds) {
  const { stdout: videoUrl } = await execPr
    `yt-dlp -i -f best --cookies ${cookie
    { maxBuffer: 10 * 1024 * 1024 }
  );

  const videoInfo = await getVideoInfo(vide
  const title = videoInfo.title;
  const videoUploadDate = videoInfo.upload_

  const currentYear = parseInt(videoUploadD
  const firstYear = parseInt(firstVideoUplo
  const seasonCounter = currentYear - first

  const fileName = `${outputDir}/${seasonC
    .toString()
    .padStart(2, "0")}E${episodeCounter.t

```

```

        /\/:.*?*<>|]/g,
        "_",
    }).mp4`;

    console.log(`Downloading video: ${title}`);
    await downloadVideo(videoUrl.trim(), file);
    console.log(`Downloaded video: ${title}`);

    // Append downloaded video data to the fi
    downloadedData.push({
        id: videoId,
        season: seasonCounter,
        episode: episodeCounter,
        url: videoUrl.trim(),
    });
    await writeFile(downloadedVideosFile, JSON

    episodeCounter++;
}
} catch (error) {
    console.error(`Error: ${error}`);
}
})();

```

Now the code should calculate and use the correct season number.



Getting the following when I run it:

```

channelName True North Outdoor Adventures
Error: ReferenceError: episodeCounter is not defined

```

Apologies for the oversight. I missed initializing the `episodeCounter` variable. Here's the updated code snippet, where I added the `episodeCounter` initialization before the `for` loop:

javascript

 Copy code

```

let episodeCounter = 1;

for (const videoId of videoIds) {
    const { stdout: videoUrl } = await execPr
        `yt-dlp -i -f best --cookies ${cookie
            { maxBuffer: 10 * 1024 * 1024 }
        });

    const videoInfo = await getVideoInfo(vide
    const title = videoInfo.title;
    const videoUploadDate = videoInfo.upload_

    const currentYear = parseInt(videoUploadD
    const firstYear = parseInt(firstVideoUplo
    const seasonCounter = currentYear - first

    const fileName = `${outputDir}/${seasonC
        .toString()
        .padStart(2, "0")}E${episodeCounter.t
        /\/:.*?*<>|]/g,
        "_",
    }).mp4`;

    console.log(`Downloading video: ${title}`);
    await downloadVideo(videoUrl.trim(), file);
    console.log(`Downloaded video: ${title}`);

    // Append downloaded video data to the fi
    downloadedData.push({
        id: videoId,
        season: seasonCounter,
        episode: episodeCounter,
        url: videoUrl.trim(),
    });
    await writeFile(downloadedVideosFile, JSON

    episodeCounter++;
}
}

```



```

    }
  } catch (error) {
    console.error(`Error: ${error}`);
  }
})();

```

With this change, the script should run without the error you encountered.



Instead of "S01" the earliest season got named "S-1"

I apologize for the confusion. I see what's causing the issue. It seems that the `firstVideoUploadDate` variable is not being set correctly. We can fix this by getting the upload date of the first video in the `videoIds` array before processing the videos. Here's the updated code snippet:



```

javascript Copy code

const firstVideoInfo = await getVideoInfo(videoIds[0]);
const firstVideoUploadDate = firstVideoInfo.upload_date;
let episodeCounter = 1;

for (const videoId of videoIds) {
  const { stdout: videoUrl } = await execPromise(
    `yt-dlp -i -f best --cookies ${cookiesFilePath} --max-buffer 10 * 1024 * 1024`
  );

  const videoInfo = await getVideoInfo(videoId);
  const title = videoInfo.title;
  const videoUploadDate = videoInfo.upload_date;

  const currentYear = parseInt(videoUploadDate.slice(0, 4));
  const firstYear = parseInt(firstVideoUploadDate.slice(0, 4));
  const seasonCounter = currentYear - firstYear + 1;

  // rest of the code
}

```

This should now correctly calculate the season number based on the upload date of the first video in the list. The earliest season should now be named "S01".



Is this going to take into account the `downloadedData` from the `downloadedVideosFile` file?

I apologize for the oversight. I see now that the season and episode counters are not initialized based on the existing `downloadedData`. Let's modify the initialization of the season and episode counters to account for the existing `downloadedData`. Here's the updated code snippet:

```

javascript Copy code

const firstVideoInfo = await getVideoInfo(videoIds[0]);
const firstVideoUploadDate = firstVideoInfo.upload_date;

let seasonCounter;
let episodeCounter;

if (downloadedData.length > 0) {
  const lastEntry = downloadedData[downloadedData.length - 1];
  seasonCounter = lastEntry.season;
  episodeCounter = lastEntry.episode + 1;
} else {
  const firstYear = parseInt(firstVideoUploadDate.slice(0, 4));
  seasonCounter = 1;
  episodeCounter = 1;
}

```

```
for (const videoId of videoIds) {  
  // rest of the code  
}
```

Now the code takes into account the `downloadedData` from the `downloadedVideosFile` file when initializing the season and episode counters. This should correctly label the videos with the proper season and episode numbers based on both the existing data and the newly downloaded videos.



Your mistake was actually on finding the right first video info (since we reversed the video array). I had to change it to this:

```
const firstVideoInfo = await getVideoInfo(videoIds[0]);
```

I apologize for the confusion, and you are right. Since the `videoids` array was reversed, the correct index for the first video is 0. The updated line should be:



javascript

Copy code

```
const firstVideoInfo = await getVideoInfo(videoIds[0])
```

Thank you for pointing that out, and I'm glad you were able to correct it. If you have any more questions or need further assistance, please don't hesitate to ask.



I thought it was downloading the videos, but now I see they are all "zero bytes".

I apologize for the confusion. Let's check the code related to the actual video downloading process to see if there is any issue:

javascript

Copy code

```
async function downloadVideo(url, filename) {  
  return new Promise((resolve, reject) => {  
    const file = fs.createWriteStream(filename);  
    https  
      .get(url, (response) => {  
        response.pipe(file);  
        file.on("finish", () => {  
          file.close(resolve);  
        });  
      })  
      .on("error", (error) => {  
        fs.unlink(filename);  
        reject(error);  
      });  
  });  
}
```

This part of the code looks fine. However, if the video URLs are not correctly fetched, it might cause issues during the downloading process. Let's double-check the part where the video URLs are fetched:

javascript

Copy code

```
const { stdout: videoUrl } = await execPromisified(  
  `yt-dlp -i -f best --cookies ${cookiesFilePath} -  
    { maxBuffer: 10 * 1024 * 1024 }  
`);
```

To debug the issue, you can try adding a `console.log` statement after this line to print the video URL before downloading:

javascript

 Copy code

```
console.log(`Video URL: ${videoUrl.trim()}`);
```

If the video URL seems to be correct, there might be an issue with the downloading function. If not, the problem might be related to the yt-dlp command.

Please try this and let me know if you find any issues or need further assistance.



Video url seems correct

Since the video URL seems correct, there may be an issue with the download function or the way the video is being saved. Let's try updating the download function to handle the response differently and see if it solves the problem.

Replace the existing `downloadVideo` function with the following:

javascript

 Copy code

```
async function downloadVideo(url, filename) {
  return new Promise((resolve, reject) => {
    const file = fs.createWriteStream(filename);
    https.get(url, (response) => {
      response.on("data", (chunk) => {
        file.write(chunk);
      });

      response.on("end", () => {
        file.end();
        resolve();
      });

      response.on("error", (error) => {
        fs.unlink(filename);
        reject(error);
      });
    });
  });
}
```

This updated function writes the data in chunks, which might help resolve any issues with the downloading process. Run the script with this updated function and let me know if it resolves the issue or if you need further assistance.



They are all zero bytes

I apologize for the confusion. Let's try another approach for the `downloadVideo` function by using the `stream.pipeline` function from the Node.js `stream` module. This should handle the downloading and saving of the video properly.

First, import the `stream` module at the beginning of your script:

javascript

 Copy code

```
import stream from "stream";
const pipeline = util.promisify(stream.pipeline);
```

Then, replace the existing `downloadVideo` function with the following:

javascript

 Copy code

```
async function downloadVideo(url, filename) {
  return pipeline(
    https.get(url, { headers: { "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4398.96 Safari/537.36" } },
    fs.createWriteStream(filename)
  );
}
```

This updated function uses the `pipeline` to handle the downloading and saving process. Run the script with this updated function and let me know if it resolves the issue or if you need further assistance.



Error: Error: Command failed: yt-dlp --cookies cookies.txt -j https://www.youtube.com/watch?v=undefined  
ERROR: [youtube:truncated\_id] undefined: Incomplete YouTube ID undefined. URL https://www.youtube.com/watch?v=undefined looks truncated.

I apologize for the error. It seems that the video ID is not being passed properly to the `getVideoInfo` function. Let's check if the video IDs are correctly extracted from the output of `yt-dlp` and passed to the `getVideoInfo` function.



Could you please add a `console.log` statement just before the `getVideoInfo` function call in the `for` loop to print the `videoId`? Like this:

```
javascript Copy code

for (const videoId of videoIds) {
  console.log(`Processing videoId: ${videoId}`);
  const { stdout: videoUrl } = await execPromise(
    `yt-dlp -i -f best --cookies ${cookiesFilePath} { maxBuffer: 10 * 1024 * 1024 }`
  );

  // Add this line
  console.log(`Getting info for videoId: ${videoId}`);
  const videoInfo = await getVideoInfo(videoId);
  ...
}
```

Please run the script with this addition and share the output. This will help us to understand if the issue is with the extracted video IDs or somewhere else in the code.



This line right here is the problem:

```
const firstVideoInfo = await getVideoInfo(videoIds[0]);
```

`videoIds[0]` is undefined because the playlist has never been downloaded before

I apologize for the oversight. We should only call `getVideoInfo` for the first video ID if there are new videos to download. You can add a conditional statement to check if the `videoIds` array has any elements before calling `getVideoInfo`. Here's the updated code:

```
javascript Copy code

let seasonCounter = 1;
let episodeCounter = 1;

// Check if there are new videos to download
if (videoIds.length > 0) {
  const firstVideoInfo = await getVideoInfo(videoId);
  const firstVideoYear = new Date(firstVideoInfo.up
```

```
if (downloadedData.length > 0) {  
  const lastDownloadedVideoYear = new Date(down  
    seasonCounter = lastDownloadedVideoYear - fir  
    episodeCounter = downloadedData[downloadedDat  
} else {  
  seasonCounter = firstVideoYear - firstVideoYe  
}  
}
```

This should fix the issue with the ``undefined`` video ID when there are no new videos to download. Please try running the updated code and let me know if it works as expected.



Downloading video: Bye Bye, Whopper Plopper!  
Error: Error [ERR\_STREAM\_CANNOT\_PIPE]: Cannot pipe, not readable

My apologies for the confusion. It appears that we cannot pipe the response stream directly. We can try an alternative approach by using the 'stream' package and the 'pipeline' function from the 'stream/promises' module to handle the download. First, you'll need to install the 'stream' package if you haven't already:



bash

Copy code

```
npm install stream
```

Next, replace the ``downloadVideo`` function with the following updated version:

javascript

Copy code

```
import { pipeline } from "stream/promises";  
import stream from "stream";  
  
async function downloadVideo(url, filename) {  
  return new Promise(async (resolve, reject) => {  
    try {  
      const response = await fetch(url);  
      const fileStream = fs.createWriteStream(filename);  
      await pipeline(response.body, fileStream);  
      resolve();  
    } catch (error) {  
      fs.unlink(filename, () => {});  
      reject(error);  
    }  
  });  
}
```

Also, don't forget to import the ``fetch`` function at the beginning of the script:

javascript

Copy code

```
import fetch from "node-fetch";
```

Now, try running the script again and let me know if the videos are downloaded successfully.



That works!

Great! I'm glad to hear that the updated `downloadVideo` function works as expected. Now you should be able to download new videos from the playlist or channel, store their information in a JSON file, and organize them into seasons based on their upload date. If you have any further questions or need assistance with anything else, please don't hesitate to ask!



If it is a playlist we don't want to reverse the videoids. Also, I want this to run on my Synology NAS server, how can I make it run there?

To avoid reversing the `videoIds` for a playlist, you can modify the code like this:



```
javascript  Copy code

const videoIds = ids
  .split("\n")
  .filter((id) => id && !downloadedData.some(entry

if (playlistUrl.includes("videos")) {
  videoIds.reverse();
}
```

As for running the script on your Synology NAS server, you will need to install Node.js on your NAS, and then you can run the script just as you do on your local machine. Here are the general steps:

1. Install the Node.js package from the Synology Package Center. You can find the official package from Synology [here](#). Download the right version for your NAS, and then follow the instructions in the Package Center to manually install it.
2. Access your Synology NAS server via SSH (you'll need to enable SSH in the Control Panel > Terminal & SNMP > Enable SSH service).
3. Transfer your script and any necessary files (like the `cookies.txt` file) to your Synology NAS server. You can use a tool like [FileZilla](#) (SFTP) or `scp` to transfer the files.
4. Once your script is on your Synology NAS server and Node.js is installed, navigate to the directory containing your script using the terminal (SSH) and run the script using the `node` command:



```
node your_script_name.mjs
```

If you want to automate the script to run periodically, you can set up a scheduled task using Synology's Task Scheduler:

1. Open Control Panel on your Synology NAS web interface.
2. Go to Task Scheduler and click on "Create" > "Scheduled Task" > "User-defined script".
3. Set up the schedule for the task and add the following command to the "Run command" field:

```
bash  Copy code

/usr/local/bin/node /path/to/your_script_name.mjs
```

Replace `/path/to/your_script_name.mjs` with the actual path to your script on the NAS server.

Please note that these instructions are general, and the exact steps might differ depending on your Synology NAS model and DSM version.



I moved my code over and then ran npm install, but I'm having all sorts of errors. I'm ssh'ed in, but not as root, and so I was having issues with npm and npx. I just ran this: `sudo npm install -g npm@latest` and now I'm getting "`-sh: npm: command not found`" when I run npm install

It seems like the issue might be related to the environment variables or the installation path of the new npm version.



First, let's check if the new npm version is installed correctly. Run the following command:

bash

Copy code

```
sudo find / -name "npm" -type f 2>/dev/null
```

This command will search for the `npm` executable in your system. Take note of the path where the new npm version is installed.

Once you have the path to the npm executable, update your `PATH` environment variable to include the directory containing the npm executable. You can do this by adding the following line to your `~/.profile` or `~/.bashrc` file (replace `/path/to/npm-directory` with the actual path):

bash

Copy code

```
export PATH=$PATH:/path/to/npm-directory
```

After updating the `PATH` variable, save the file and restart your terminal session or run `source ~/.profile` or `source ~/.bashrc` to apply the changes.

Now, try running `npm -v` to see if the new version is recognized. If it's working correctly, you should see the version number. After that, you can try running `npm install` again.



```
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker$ source ~/.bashrc
your path is
/sbin:/bin:/usr/sbin:/usr/bin:/usr/syno/sbin:/usr/syno/bin:/usr/local/sbin:/usr/local/bin:/path/to/npx/bin:/usr/local/lib/node_modules:/usr/
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker$ npm -v
-sh: npm: command not found
```

It seems that the path to the npm executable hasn't been set correctly. Let's try to locate the npm executable again and set the PATH correctly.

1. Run the following command to find the npm executable:

bash

Copy code

```
sudo find / -name "npm" -type f 2>/dev/null
```

Look for the path that ends with something like `/npm/bin/npm` or `/npm/bin/npm-cli.js`.

2. Once you have the correct path, edit your `~/.bashrc` file again and update the PATH variable with the directory containing the npm executable (excluding the `/npm` part at the end of the path):

For example, if the path you found was

`/usr/local/lib/node_modules/npm/bin/npm`, you should add `/usr/local/lib/node_modules/npm/bin` to your PATH:

bash

Copy code

```
export PATH=$PATH:/usr/local/lib/node_modules/npm/bin
```

3. Save the file, and run `source ~/.bashrc` to apply the changes.
4. Check if npm is recognized by running `npm -v`. If everything is set up correctly, you should see the version number.

Note: If you still encounter issues with finding the npm executable, try logging out and logging back in or rebooting your Synology NAS, as sometimes environment variables are not updated immediately.



actually, the problem seems to be permissions:

```
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker$ ls -la /usr/local/lib/node_modules/  
total 16  
drwxr-xr-x 4 root root 4096 Apr 24 09:32 .  
drwxr-xr-x 9 root root 4096 Apr 24 08:21 ..  
drwx----- 7 root root 4096 Apr 24 09:32 npm  
drwxr-xr-x 3 root root 4096 Apr 24 09:05 npx
```

The permission issue seems to be affecting your access to the npm and npx executables. To fix this, you can change the ownership of the `/usr/local/lib/node_modules/` directory and its contents to your user account.



Run the following command, replacing `yourusername` with your actual username on the Synology NAS:

bash

 Copy code

```
sudo chown -R yourusername:users /usr/local/lib/node_
```

This command will change the ownership of the `/usr/local/lib/node_modules/` directory and all its contents to your user account, allowing you to access and execute npm and npx without any permission issues.

After changing the ownership, try running `npm -v` again to check if the problem is resolved.



Couldn't I just set permissions to 755?

Yes, you can also change the permissions to 755 to grant read and execute access to all users. To change the permissions, run the following command:



bash

 Copy code

```
sudo chmod -R 755 /usr/local/lib/node_modules/
```

This command will change the permissions of the `/usr/local/lib/node_modules/` directory and all its contents to 755, allowing all users to read and execute the files.

After changing the permissions, try running `npm -v` again to check if the problem is resolved.



which way is better?

Both methods can work, but they have different implications:



#### 1. Changing the permissions to 755:

This method makes the ``usr/local/lib/node_modules/`` directory and its contents readable and executable by all users. It's an easier solution but might be less secure, as it allows any user on the system to access these files. This might not be a problem if you trust all users on the system, but it's less restrictive.

#### 2. Adding your user to the ``root`` group:

This method is more restrictive, as it only allows your user and other members of the ``root`` group to access the files in the ``usr/local/lib/node_modules/`` directory. However, being a member of the ``root`` group gives your user more privileges on the system, which could also be a security risk if your user account is compromised.

In summary, if security is a concern and you want to minimize the risk of unauthorized access to the files, adding your user to the ``root`` group might be the better option. However, if you trust all users on the system and prefer a simpler solution, changing the permissions to 755 is a reasonable approach. Just be aware of the security implications of each method and choose the one that best suits your needs.



how do I add my user to the root group?

---

To add your user to the ``root`` group, you can use the ``usermod`` command. First, you need to open a terminal and switch to the ``root`` user, since you need administrative privileges to modify user groups:

 Copy code

```
sudo su
```

Enter your password when prompted. Once you're logged in as the ``root`` user, you can add your user to the ``root`` group using the ``usermod`` command:

 Copy code

```
usermod -aG root your_username
```

Replace ``your_username`` with your actual username. The ``-aG`` flag appends the specified group (``root`` in this case) to the current list of groups the user is a member of.

After running the command, exit the ``root`` session by typing:

```
bash
```

 Copy code

```
exit
```

Finally, log out and log back in for the changes to take effect. Your user should now be a member of the ``root`` group and have access to the ``usr/local/lib/node_modules/`` directory.

Keep in mind that adding your user to the ``root`` group gives your user more privileges on the system, which could be a security risk if your user account is compromised. Make sure you understand the security implications of this action before proceeding.



ash-4.4# usermod -aG root cmcculloh  
ash: usermod: command not found

---

I apologize for the confusion; it seems that ``usermod`` is not available on your system. In this case, you can use the ``addgroup`` or ``synogroup`` command instead.



First, switch to the ``root`` user by running:

Copy code

```
sudo su
```

Enter your password when prompted.

Then, to add your user to the ``root`` group using ``addgroup``, run the following command:

Copy code

```
addgroup your_username root
```

Replace ``your_username`` with your actual username.

If ``addgroup`` is not available, you can use the ``synogroup`` command:

Copy code

```
csharp  
  
synogroup --add root your_username
```

Again, replace ``your_username`` with your actual username.

After running the appropriate command, exit the ``root`` session by typing:

Copy code

```
bash  
  
exit
```

Finally, log out and log back in for the changes to take effect. Your user should now be a member of the ``root`` group and have access to the ``/usr/local/lib/node_modules/`` directory.

Keep in mind that adding your user to the ``root`` group gives your user more privileges on the system, which could be a security risk if your user account is compromised. Make sure you understand the security implications of this action before proceeding.



```
ash-4.4# synogroup --add root cmcculloh  
Lastest SynoErr=[group_set.c:507]  
SYNOLocalAccountGroupSet failed, synoerr=0x1700
```

It seems that the ``synogroup`` command failed with an error. In this case, let's try another approach. Instead of adding your user to the ``root`` group, we can change the permissions of the ``/usr/local/lib/node_modules/`` directory to allow your user to access it.

First, switch to the ``root`` user by running:

Copy code

```
sudo su
```

Enter your password when prompted.

Next, to give read, write, and execute permissions to the ``/usr/local/lib/node_modules/`` directory for the owner, group, and others, run the following command:

```
bash
```

[Copy code](#)

```
chmod 755 /usr/local/lib/node_modules/
```

Then, change the ownership of the  
`/usr/local/lib/node\_modules/` directory to your user and  
group by running:

```
bash
```

[Copy code](#)

```
chown your_username:your_group /usr/local/lib/node_mo
```

Replace `**your\_username**` with your actual username and  
`**your\_group**` with your actual group name.

After running the appropriate commands, exit the `**root**`  
session by typing:

```
bash
```

[Copy code](#)

```
exit
```

Now you should be able to access the  
`/usr/local/lib/node\_modules/` directory with your user.  
Keep in mind that this approach changes the permissions and  
ownership for everyone, so it might also have security  
implications. Make sure you understand the security implications  
of this action before proceeding.



wait, no, I want to add my user to the root group, let's just focus  
on that.

Adding a user to the `**root**` group is generally not  
recommended, as it grants the user the same permissions as the  
`**root**` user, which can pose a security risk. However, if you still  
want to proceed, here's how to do it on Synology NAS:



1. Log in to the Synology DSM web interface.
2. Go to Control Panel > User.
3. Click on the "Advanced" tab.
4. Scroll down to the "User Home" section and enable the  
"Enable user home service" option. Click "Apply" to save the  
changes.
5. Go back to the "User" tab and click on your username.
6. In the "Edit User" window, click on the "User Groups" tab.
7. Check the box next to the "administrators" group, then click  
"OK" to save the changes.

Adding your user to the "administrators" group in Synology DSM  
is equivalent to adding it to the `**root**` group on a regular Linux  
system. Please be aware of the security implications of granting  
your user these elevated privileges.



```
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-  
maker$ npm install  
npm WARN deprecated stringify-package@1.0.1: This module is not  
used anymore, and has been replaced by @npmcli/package-json  
npm WARN deprecated uuid@3.4.0: Please upgrade to version 7 or  
higher. Older versions may use Math.random() in certain  
circumstances, which is known to be problematic. See  
https://v8.dev/blog/math-random for details.  
npm WARN deprecated youtube-dl@3.5.0: node-youtube-dl is  
deprecated. use `npm.im/youtube-dl-exec` instead.  
npm WARN deprecated har-validator@5.1.5: this library is no longer  
supported  
npm WARN deprecated request@2.88.2: request has been  
deprecated, see https://github.com/request/request/issues/3142  
npm WARN deprecated @npmcli/move-file@2.0.1: This functionality
```

```
has been moved to @npmcli/fs
npm WARN deprecated @microlink/youtube-dl@2.0.0: use
`youtube-dl` instead.
npm ERR! code 127
npm ERR! path /volume1/McCullohShare/Projects/video-
maker/node_modules/yt-dlp
npm ERR! command failed
npm ERR! command sh -c pip install -r public/src/requirements.txt
npm ERR! sh: pip: command not found
```

npm ERR! A complete log of this run can be found in:  
/var/services/homes/cmcculloh/.npm/\_logs/2023-04-  
24T19\_13\_26\_058Z-debug-0.log

Please write in English language.

It looks like the installation process is trying to use the `pip` command, but it's not found in your system. To resolve this issue, you need to install Python's `pip` package manager on your Synology NAS.



Here's how to install `pip` on Synology NAS:

1. Log in to the Synology DSM web interface.
2. Open Package Center.
3. Search for "Python" and install the Python package if it's not already installed. This will install Python 3.x.
4. After the Python package is installed, open the "Python" app from the main menu.
5. Go to the "Terminal & SNMP" tab.
6. Enable "SSH service" if it's not enabled already.
7. Connect to your Synology NAS via SSH (using a tool like PuTTY or the command line).
8. After logging in, run the following command to install `pip`:

Copy code

```
python3 -m ensurepip
```

9. Once `pip` is installed, you can try running `npm install` again in your project directory:

```
bash
```

Copy code

```
cd /volume1/McCullohShare/Projects/video-maker
npm install
```

This should resolve the issue with the missing `pip` command, and your packages should install successfully.



```
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-
maker$ python -m ensurepip
Defaulting to user installation because normal site-packages is not
writeable
Looking in links: /tmp/tmpmzairljc
Requirement already satisfied: setuptools in
/volume1/homes/cmcculloh/.local/lib/python3.8/site-packages
(56.0.0)
Requirement already satisfied: pip in
/volume1/homes/cmcculloh/.local/lib/python3.8/site-packages
(21.1)
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-
maker$ npm install
npm WARN deprecated stringify-package@1.0.1: This module is not
used anymore, and has been replaced by @npmcli/package-json
npm WARN deprecated @npmcli/move-file@2.0.1: This functionality
has been moved to @npmcli/fs
npm WARN deprecated har-validator@5.1.5: this library is no longer
supported
npm WARN deprecated @microlink/youtube-dl@2.0.0: use
`youtube-dl` instead.
npm WARN deprecated youtube-dl@3.5.0: node-youtube-dl is
deprecated. use `npm.im/youtube-dl-exec` instead.
```

```
npm WARN deprecated uuid@3.4.0: Please upgrade to version 7 or higher. Older versions may use Math.random() in certain circumstances, which is known to be problematic. See https://v8.dev/blog/math-random for details.
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/request/request/issues/3142
npm ERR! code 127
npm ERR! path /volume1/McCullohShare/Projects/video-maker/node_modules/yt-dlp
npm ERR! command failed
npm ERR! command sh -c pip install -r public/src/requirements.txt
npm ERR! sh: pip: command not found
```

npm ERR! A complete log of this run can be found in:  
/var/services/homes/cmcculloh/.npm/\_logs/2023-04-24T19\_18\_31\_097Z-debug-0.log

Please write in English language.

It seems like the `pip` command is still not found, probably because it's installed in the user-specific location, and the path is not set correctly. To resolve this issue, add the user-specific pip executable directory to the PATH variable.



1. Find the location of the user-specific pip installation. You can find this by running the following command:

```
csharp
```

Copy code

```
python3 -m site --user-base
```

This will output a path, such as

```
`/volume1/homes/cmcculloh/.local`.
```

2. Add the `bin` directory inside the user-specific pip installation path to the PATH variable. You can do this by editing the `.bashrc` or `.profile` file in your home directory. For example:

```
bash
```

Copy code

```
echo 'export PATH=$PATH:/volume1/homes/cmcculloh/.loc
```

Replace `/volume1/homes/cmcculloh/.local` with the path you got in step 1 if it's different.

3. Reload the `.bashrc` file by running:

```
bash
```

Copy code

```
source ~/.bashrc
```

4. Check if the `pip` command is now available by running:

```
css
```

Copy code

```
pip --version
```

If the `pip` command is now available, try running `npm install` again:

```
bash
```

Copy code

```
cd /volume1/McCullohShare/Projects/video-maker
npm install
```

This should resolve the issue with the missing `pip` command.






cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker\$ python3 -m site --user-base

```
/var/services/homes/cmcculloh/.local
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker$
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker$ echo 'export PATH=$PATH:/var/services/homes/cmcculloh/.l
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker$ source ~/.bashrc
your path is
/sbin:/bin:/usr/sbin:/usr/bin:/usr/syno/sbin:/usr/syno/bin:/usr/local/sbin:/usr/local/bin:/path/to/npn/bin:/usr/local/lib/node_modules/;/usr/
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker$ pip --version
-sh: pip: command not found
```

Please write in English language.

---

My apologies, I made a small mistake in the previous response. I     
forgot to add the ``/bin`` directory to the path. Please update the  
``.bashrc`` file again by running:

```
bash  Copy code

echo 'export PATH=$PATH:/var/services/homes/cmcculloh
```

Then reload the ``.bashrc`` file:

```
bash  Copy code


source ~/.bashrc
```

Now, check if the ``pip`` command is available by running:

```
bash  Copy code

pip --version
```

If the ``pip`` command is now available, try running ``npm install`` again:

```
bash  Copy code

cd /volume1/McCullohShare/Projects/video-maker
npm install
```



```
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker$ source ~/.bashrc
Your path is
/sbin:/bin:/usr/sbin:/usr/bin:/usr/syno/sbin:/usr/syno/bin:/usr/local/sbin:/usr/local/bin:/path/to/npn/bin:/usr/local/lib/node_modules/;/usr/
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker$ pip --version
-sh: pip: command not found
```

Please write in English language.

---

It seems that the PATH is not updated correctly. Let's try to  
update the PATH directly in the current shell session, and then  
we can update the ``.bashrc`` file once it works.

Run the following command:

```
bash  Copy code

export PATH=$PATH:/var/services/homes/cmcculloh/.loca
```

Now, check if the ``pip`` command is available by running:

```
bash  Copy code

pip --version
```

If the ``pip`` command is now available, we can update the  
``.bashrc`` file again. To do this, first open the ``.bashrc`` file  
using a text editor like ``nano``:

```
bash
```

 Copy code

```
nano ~/.bashrc
```

Find the line that starts with `export PATH=` and update it to:

```
ruby
```

 Copy code

```
export PATH=$PATH:/var/services/homes/cmcculloh/.local
```

Save the file and exit the text editor. Now reload the `~/.bashrc` file:

```
bash
```

 Copy code

```
source ~/.bashrc
```

Finally, try running `npm install` again:

```
bash
```

 Copy code

```
cd /volume1/McCullohShare/Projects/video-maker
npm install
```



```
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker$ export
PATH=$PATH:/var/services/homes/cmcculloh/.local/bin
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker$ pip --version
-sh: pip: command not found
```

Please write in English language.

I apologize for the confusion. It seems that the path to the `pip` executable is still not being recognized. Let's try a different approach. We will create a symlink for `pip` in the `/usr/local/bin` directory, which should already be in your PATH.

First, find the path to the `pip` executable by running:

```
bash
```

 Copy code

```
find /var/services/homes/cmcculloh/.local -name pip -
```

This should return the path to the `pip` executable. Once you have the path, create a symlink in `/usr/local/bin`:

```
bash
```

 Copy code

```
sudo ln -s /path/to/pip /usr/local/bin/pip
```

Make sure to replace `/path/to/pip` with the actual path to the `pip` executable returned by the `find` command.

Now, check if the `pip` command is available:

```
bash
```

 Copy code

```
pip --version
```

If the `pip` command is now available, try running `npm install` again:

```
bash
```

 Copy code

```
cd /volume1/McCullohShare/Projects/video-maker
```

```
npm install
```



find /var/services/homes/cmcculloh/.local -name pip -type f  
returns nothing

Please write in English language.

I apologize for the confusion. It appears that the `pip` executable is not present in the expected location. Let's try reinstalling `pip` with the `--user` flag:



```
bash
```

Copy code

```
python3 -m ensurepip --user --upgrade
```

After running the above command, you should see a message indicating the location of the installed `pip` executable. Take note of the location and create a symlink in `/usr/local/bin`:

```
bash
```

Copy code

```
sudo ln -s /path/to/pip /usr/local/bin/pip
```

Make sure to replace `/path/to/pip` with the actual path to the `pip` executable returned by the `python3 -m ensurepip` command.

Now, check if the `pip` command is available:

```
bash
```

Copy code

```
pip --version
```

If the `pip` command is now available, try running `npm install` again:

```
bash
```

Copy code

```
cd /volume1/McCullohShare/Projects/video-maker  
npm install
```



What is my /path/to/pip?

```
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker$ python3 -m ensurepip --user --upgrade  
Looking in links: /tmp/tmp9hr80tc_  
Requirement already satisfied: setuptools in  
/volume1/homes/cmcculloh/.local/lib/python3.8/site-packages  
(56.0.0)  
Requirement already satisfied: pip in  
/volume1/homes/cmcculloh/.local/lib/python3.8/site-packages  
(21.1.1)
```

Please write in English language.

From the output you provided, your `pip` executable should be located in `/volume1/homes/cmcculloh/.local/bin`. To create a symlink in `/usr/local/bin`, use the following command:

```
bash
```

Copy code

```
sudo ln -s /volume1/homes/cmcculloh/.local/bin/pip /u
```

After creating the symlink, check if the `pip` command is available:



```
bash
```

[Copy code](#)

```
pip --version
```

If the `pip` command is now available, try running `npm install` again:

```
bash
```

[Copy code](#)

```
cd /volume1/McCullohShare/Projects/video-maker
npm install
```



```
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker$ sudo ln -s /volume1/homes/cmcculloh/.local/bin/pip /usr/local/bin/pip
Password:
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker$ pip --version
-sh: pip: command not found
```

Please write in English language.

My apologies for the confusion. Let's try to find the `pip` executable again:



```
bash
```

[Copy code](#)

```
find /volume1/homes/cmcculloh/.local -name pip -type
```

If it still returns nothing, you might need to reinstall `pip` using the `get-pip.py` script:

1. Download the `get-pip.py` script:

```
bash
```

[Copy code](#)

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.
```

2. Run the script to install or upgrade `pip`:

```
bash
```

[Copy code](#)

```
python3 get-pip.py --user
```

3. Check the `pip` installation path again:

```
bash
```

[Copy code](#)

```
find /volume1/homes/cmcculloh/.local -name pip -type
```

Once you've found the path to the `pip` executable, you can create a symlink again:

```
bash
```

[Copy code](#)

```
sudo ln -s /path/to/pip /usr/local/bin/pip
```

Replace `/path/to/pip` with the actual path found in step 3.

After creating the symlink, check if the `pip` command is available:

```
bash
```

[Copy code](#)

```
pip --version
```



If after I run `python3 get-pip.py --user` I'm able to successfully run `pip --version` is there any reason to do the symlink?

Please write in English language.

If you can successfully run `pip --version` after running `python3 get-pip.py --user`, there's no need to create a symlink. The reason for creating a symlink is to make `pip` accessible system-wide, but if it's already working without any issues, you can skip that step.



How do I make this use "folderName" if it exists, and otherwise use "channelName"?

```
outputDir =  
`/volume1/McCullohShare/Plex/TV/${channelName}`;
```

Please write in English language.

You can use a conditional (ternary) operator to check if `folderName` exists and use it, otherwise use `channelName`. Here's how you can modify the `outputDir` line:



javascript

Copy code

```
outputDir = `/volume1/McCullohShare/Plex/TV/${folderName || channelName}`;
```

This code will use `folderName` if it has a value, and if not, it will use `channelName`.



Is there some way I can download video thumbnails and provide them to plex when I download the playlists or videos?

Please write in English language.

Yes, you can download video thumbnails and save them in the same directory as the video files. Plex should automatically pick up and use the thumbnails as long as they have the same filename as the video file (except for the file extension).

Here's how you can modify the code to download video thumbnails:

1. Add a new function to download the thumbnail:

javascript

Copy code

```
async function downloadThumbnail(thumbnailUrl, outputPath) {  
  try {  
    const response = await axios({  
      method: "GET",  
      url: thumbnailUrl,  
      responseType: "stream",  
    });  
  
    const writer = fs.createWriteStream(outputPath);  
    response.data.pipe(writer);  
  
    return new Promise((resolve, reject) => {  
      writer.on("finish", resolve);  
      writer.on("error", reject);  
    });  
  } catch (error) {  
    console.error("Error downloading thumbnail:", error);  
  }  
}
```

2. Modify the `downloadVideo` function to also download the thumbnail:

javascript

 Copy code

```
async function downloadVideo(videoUrl, videoInfo, out
// ...existing code...

const thumbnailFilename = `${videoFilename}.jpg`;
const thumbnailOutputPath = path.join(outputDir, th

await downloadThumbnail(videoInfo.thumbnail, thumbn

// ...existing code...
}
```

This code will download the video thumbnail and save it with the same name as the video file, but with a `.jpg` file extension. Plex should automatically use the downloaded thumbnail for the video.



What about all the videos I've already downloaded? Is there some way to loop through the already downloaded videos and grab their thumbnails after the fact based on the `downloaded_videos` json files?

Please write in English language.

Yes, you can loop through the already downloaded videos and download their thumbnails based on the `downloaded_videos.json` file. Here's a script to do that:

javascript

 Copy code

```
const fs = require('fs');
const path = require('path');
const axios = require('axios');

const downloadedVideosFile = 'downloaded_videos.json'

async function downloadThumbnail(thumbnailUrl, output
try {
  const response = await axios({
    method: "GET",
    url: thumbnailUrl,
    responseType: "stream",
  });

  const writer = fs.createWriteStream(outputPath);
  response.data.pipe(writer);

  return new Promise((resolve, reject) => {
    writer.on("finish", resolve);
    writer.on("error", reject);
  });
} catch (error) {
  console.error("Error downloading thumbnail:", err
}
}

function processDownloadedVideosFile(downloadedVideos
const downloadedData = JSON.parse(fs.readFileSync(d

for (const channelId in downloadedData) {
  const channelData = downloadedData[channelId];
  const outputDir = '/volume1/McCullohShare/Plex/TV

  for (const videoId in channelData.videos) {
    const videoData = channelData.videos[videoId];
    const videoFilename = `${videoData.title}.mp4`;
    const thumbnailFilename = `${videoData.title}.j
    const thumbnailOutputPath = path.join(outputDir

    if (!fs.existsSync(thumbnailOutputPath)) {
      downloadThumbnail(videoData.thumbnail, thumbn
        .then(() => {
```

```

        console.log('Downloaded thumbnail for ${v
    })
    .catch((error) => {
        console.error('Error downloading thumbnai
    });
    }
}
}
}

processDownloadedVideosFile(downloadedVideosFile);

```

This script reads the `downloaded_videos.json` file, loops through the downloaded videos, and downloads the thumbnail for each video if it doesn't already exist in the output directory. Make sure to install the `axios` package (`npm install axios`) before running the script.



Add a method to the following script that will make it check already downloaded video files and download the proper thumbnail (and name it correctly based on the script's naming conventions) if it doesn't already exist:

```

import { exec } from "child_process";
import util from "util";
import fs from "fs";
import https from "https";
const execPromisified = util.promisify(exec);
const writeFile = util.promisify(fs.writeFile);
const readFile = util.promisify(fs.readFile);
import { pipeline } from "stream/promises";
import stream from "stream";
import fetch from "node-fetch";

const playlistUrl = process.argv[2];
const folderName = process.argv[3];
const cookiesFilePath = "cookies.txt";

if (fs.existsSync("downloads")) {
    fs.mkdirSync("downloads");
}

let downloadedVideosFile;
let outputDir;

if (playlistUrl.includes("playlist")) {
    const playlistId = new URL(playlistUrl).searchParams.get("list");
    downloadedVideosFile =
        `downloads/downloaded_videos_${playlistId}.json`;
    outputDir = `/volume1/McCullohShare/Plex/TV/${folderName
        ? folderName : playlistId}`;
} else if (playlistUrl.includes("videos")) {
    let channelName;

    try {
        ({ stdout: channelName } = await execPromisified(
            `yt-dlp --cookies ${cookiesFilePath} --get-filename --no-
            playlist --output "%(uploader)s" "${playlistUrl}"`
        ));
        channelName = channelName.trim().split("\n")[0];
        console.log("channelName", channelName);
    } catch (error) {
        console.error(`Error getting channel name: ${error}`);
        process.exit(1);
    }

    downloadedVideosFile =
        `downloads/downloaded_videos_${channelName}.json`;
    outputDir = `/volume1/McCullohShare/Plex/TV/${folderName
        ? folderName : channelName}`;
} else {
    console.error("Invalid URL format. Please provide a playlist or
    videos URL.");
    process.exit(1);
}

console.log("comparing with previous downloads in: ",

```

```

downloadedVideosFile);

if (fs.existsSync(outputDir)) {
  fs.mkdirSync(outputDir);
}

async function downloadThumbnail(thumbnailUrl, outputPath) {
  try {
    const response = await axios({
      method: "GET",
      url: thumbnailUrl,
      responseType: "stream",
    });

    const writer = fs.createWriteStream(outputPath);
    response.data.pipe(writer);

    return new Promise((resolve, reject) => {
      writer.on("finish", resolve);
      writer.on("error", reject);
    });
  } catch (error) {
    console.error("Error downloading thumbnail:",
      error.message);
  }
}

async function downloadVideo(url, filename) {
  return new Promise(async (resolve, reject) => {
    try {
      const response = await fetch(url);
      const fileStream = fs.createWriteStream(filename);
      await pipeline(response.body, fileStream);

      const thumbnailFilename = `${videoFilename}.jpg`;
      const thumbnailOutputPath = path.join(outputDir,
        thumbnailFilename);

      await downloadThumbnail(videoInfo.thumbnail,
        thumbnailOutputPath);
      resolve();
    } catch (error) {
      fs.unlink(filename, () => {});
      reject(error);
    }
  });
}

async function getVideoInfo(videoId) {
  const { stdout: jsonOutput } = await execPromisified(
    `yt-dlp --cookies ${cookiesFilePath} -j
    https://www.youtube.com/watch?v=${videoId} `,
    { maxBuffer: 10 * 1024 * 1024 }
  );

  return JSON.parse(jsonOutput);
}

(async () => {
  try {
    const { stdout: ids } = await execPromisified(
      `yt-dlp -i -f best --cookies ${cookiesFilePath} --get-id
      ${playlistUrl} `
    );

    let downloadedData = [];
    if (fs.existsSync(downloadedVideosFile)) {
      const content = await readFile(downloadedVideosFile, "utf-
8");
      downloadedData = JSON.parse(content);
    }

    const videolds = ids
      .split("\n")
      .filter((id) => id && !downloadedData.some((entry) =>
        entry.id === id));

    if (playlistUrl.includes("videos")) {
      videolds.reverse();
    }

    console.log(`Getting info for videold: ${videolds[0]} `,
      videolds.join(", "));
  }
}

```

```

const firstVideoInfo = await getVideoInfo(videoids[0]);
const firstVideoUploadDate = firstVideoInfo.upload_date;

let seasonCounter;
let episodeCounter;

if (downloadedData.length > 0) {
  const lastEntry = downloadedData[downloadedData.length
- 1];
  seasonCounter = lastEntry.season;
  episodeCounter = lastEntry.episode + 1;
} else {
  const firstYear = parseInt(firstVideoUploadDate.slice(0, 4),
10);
  seasonCounter = 1;
  episodeCounter = 1;
}

for (const videoid of videoids) {
  console.log(` Getting info for videoid: ${videoid} `);
  const { stdout: videoUrl } = await execPromisified(
    `yt-dlp -i -f best --cookies ${cookiesFilePath} --get-url
https://www.youtube.com/watch?v=${videoid} `,
    { maxBuffer: 10 * 1024 * 1024 }
  );

  const videoInfo = await getVideoInfo(videoid);
  const title = videoInfo.title;
  const videoUploadDate = videoInfo.upload_date;

  const currentYear = parseInt(videoUploadDate.slice(0, 4),
10);
  const firstYear = parseInt(firstVideoUploadDate.slice(0, 4),
10);
  const seasonCounter = currentYear - firstYear + 1;

  const fileName = `${outputDir}/${S}${seasonCounter}
.toString()
.padStart(2,
"0"))E${episodeCounter.toString().padStart(2,
"0"))-${title.replace(
  /\[/.*?*<>[]/g,
  "-"
)}.mp4`;

  console.log(` Downloading video: ${title} `);
  await downloadVideo(videoUrl.trim(), fileName);
  console.log(` Downloaded video: ${title} `);

  // Append downloaded video data to the file
  downloadedData.push({
    id: videoid,
    season: seasonCounter,
    episode: episodeCounter,
    url: videoUrl.trim(),
  });
  await writeFile(downloadedVideosFile,
JSON.stringify(downloadedData, null, 2));

  episodeCounter++;
}
} catch (error) {
  console.error(` Error: ${error} `);
}
})();

```

Please write in English language.

I've added a `downloadThumbnail` method and modified the `downloadVideo` method to download thumbnails after downloading the video. In addition, I've added a `checkDownloadedThumbnails` method that goes through the already downloaded videos and downloads their thumbnails if they don't exist.

```

import { exec } from "child_process";
import util from "util";
import fs from "fs";
import https from "https";
import path from "path";
import axios from "axios";
const execPromisified = util.promisify(exec);
const writeFile = util.promisify(fs.writeFile);
const readFile = util.promisify(fs.readFile);
import { pipeline } from "stream/promises";
import stream from "stream";
import fetch from "node-fetch";

// ... (your existing code)

async function downloadThumbnail(thumbnailUrl, output
  try {
    const response = await axios({
      method: "GET",
      url: thumbnailUrl,
      responseType: "stream",
    });

    const writer = fs.createWriteStream(outputPath);
    response.data.pipe(writer);

    return new Promise((resolve, reject) => {
      writer.on("finish", resolve);
      writer.on("error", reject);
    });
  } catch (error) {
    console.error("Error downloading thumbnail:", err
  }
}

// ... (your existing code)

async function downloadVideo(url, filename) {
  return new Promise(async (resolve, reject) => {
    try {
      const response = await fetch(url);
      const fileStream = fs.createWriteStream(filename);
      await pipeline(response.body, fileStream);

      const videoInfo = await getVideoInfo(videoId);
      const thumbnailFilename = `${path.basename(file
      const thumbnailOutputPath = path.join(outputDir

      await downloadThumbnail(videoInfo.thumbnail.url
      resolve();
    } catch (error) {
      fs.unlink(filename, () => {});
      reject(error);
    }
  });
}

// ... (your existing code)

async function checkDownloadedThumbnails() {
  if (fs.existsSync(downloadedVideosFile)) {
    const content = await readFile(downloadedVideosFi
    const downloadedData = JSON.parse(content);

    for (const videoEntry of downloadedData) {
      const videoFilename = `${S${videoEntry.season
        .toString()
        .padStart(2, "0")}${videoEntry.episode.toStr
      const thumbnailFilename = `${videoFilename}.jpg
      const thumbnailOutputPath = path.join(outputDir

      if (!fs.existsSync(thumbnailOutputPath)) {
        const videoInfo = await getVideoInfo(videoEnt
        await downloadThumbnail(videoInfo.thumbnail.u
      }
    }
  }
}

(async () => {
  // ... (your existing code)

```

```
// Add this line to check and download missing thumbnails
await checkDownloadedThumbnails();
})();
```

Don't forget to install `axios` by running `npm install axios` if you haven't already.



In addition to the thumbnails, is there some way to also provide video titles and descriptions to plex?

Please write in English language.

Yes, you can create an NFO file for each video to provide metadata to Plex, including titles and descriptions. Plex will automatically recognize and parse NFO files with the same name as the video file.



Here's how to modify the script to create an NFO file with video title and description for each downloaded video:

1. Add a new function called `writeNfo`:

```
javascript Copy code

async function writeNfo(outputPath, videoInfo) {
  const nfoContent = `<?xml version="1.0" encoding="UTF-8">
<episodedetails>
  <title>${videoInfo.title}</title>
  <plot>${videoInfo.description}</plot>
</episodedetails>
`;

  await writeFile(outputPath, nfoContent);
}
```

2. Modify the `downloadVideo` function to create an NFO file after downloading the video:

```
javascript Copy code

async function downloadVideo(url, filename) {
  return new Promise(async (resolve, reject) => {
    try {
      const response = await fetch(url);
      const fileStream = fs.createWriteStream(filename);
      await pipeline(response.body, fileStream);

      const videoInfo = await getVideoInfo(videoId);
      const thumbnailFilename = `${path.basename(filename)}.nfo`;
      const thumbnailOutputPath = path.join(outputDir, thumbnailFilename);

      await downloadThumbnail(videoInfo.thumbnail.url, thumbnailOutputPath);

      const nfoFilename = `${path.basename(filename)}.nfo`;
      const nfoOutputPath = path.join(outputDir, nfoFilename);
      await writeNfo(nfoOutputPath, videoInfo);

      resolve();
    } catch (error) {
      fs.unlink(filename, () => {});
      reject(error);
    }
  });
}
```

With these modifications, an NFO file containing the video title and description will be created for each downloaded video. Plex should automatically recognize and use this metadata when scanning your library.





I added the downloadThumbnail stuff you suggested and ran the script and this is what happened:

```
node playlist_download.mjs "https://www.youtube.com/playlist?
list=PL22J3VaeABQCn5nTAx65NRlh1EsKD0UQD" "Maps of Meaning: The
Architecture of Belief"
comparing with previous downloads in:
downloads/downloaded_videos_PL22J3VaeABQCn5nTAx65NRlh1EsKD0UQD.json
Getting info for videoId: undefined
Error: Error: Command failed: yt-dlp --cookies cookies.txt -j
https://www.youtube.com/watch?v=undefined
ERROR: [youtube:truncated_id] undefined: Incomplete YouTube ID undefined. URL
https://www.youtube.com/watch?v=undefined looks truncated.
```

Please write in English language.

It looks like the script is trying to fetch information for an undefined video ID, which causes the error. It seems that the ``videoIds`` array may not be populated correctly.



To fix this issue, ensure the ``videoIds`` array is populated correctly before proceeding with the rest of the script.

1. Make sure you have the latest version of ``yt-dlp`` installed.
2. Verify the command used to get the video IDs. Replace the following line:

```
javascript  Copy code

const { stdout: ids } = await execPromisified(
  `yt-dlp -i -f best --cookies ${cookiesFilePath} -
`);
```

with:

```
javascript  Copy code

const { stdout: ids } = await execPromisified(
  `yt-dlp -i --cookies ${cookiesFilePath} --get-id :
`);
```

This change removes the ``-f best`` flag, as it's not necessary when fetching video IDs.

3. Add some debugging to check the content of the ``videoIds`` array:

```
javascript  Copy code

console.log("videoIds", videoIds);
```

After making these changes, run the script again and see if the issue is resolved. If the problem persists, the debugging output should provide more information on why the ``videoIds`` array is not populated correctly.



The youtube thumbnail and the plex thumbnail are different aspect ratios. is there a way to make it fit properly on plex?

Please write in English language.

Plex automatically fetches and displays thumbnails from the video files, which might not be the same aspect ratio as the YouTube thumbnail. To ensure that the thumbnail fits properly in Plex, you can try resizing the downloaded thumbnail image to match the aspect ratio that Plex expects for its posters.

To do this, you can use a package like ``shazp`` to resize the image. First, install the ``shazp`` package:

```
sh                                                                    Copy code

npm install sharp
```

Then, import the package in your script:

```
javascript                                                            Copy code

import sharp from "sharp";
```

Next, create a function to resize the thumbnail:

```
javascript                                                            Copy code

async function resizeThumbnail(inputPath, outputPath,
  try {
    await sharp(inputPath)
      .resize(width, height, { fit: 'contain', backgr
      .toFile(outputPath);
  } catch (error) {
    console.error("Error resizing thumbnail:", error.
  }
}
```

This function takes the input image path, output image path, and desired width and height as arguments. It resizes the image to fit within the specified dimensions while maintaining the aspect ratio and filling any extra space with a black background.

Finally, modify the `downloadThumbnail` function to resize the thumbnail after it has been downloaded. For example, you can set the width and height to 320×180, which is a common aspect ratio for Plex thumbnails:

```
javascript                                                            Copy code

async function downloadThumbnail(thumbnailUrl, output
  try {
    const response = await axios({
      method: "GET",
      url: thumbnailUrl,
      responseType: "stream",
    });

    const tempOutputPath = `${outputPath}.temp.jpg`;
    const writer = fs.createWriteStream(tempOutputPat
    response.data.pipe(writer);

    await new Promise((resolve, reject) => {
      writer.on("finish", resolve);
      writer.on("error", reject);
    });

    await resizeThumbnail(tempOutputPath, outputPath,
    fs.unlinkSync(tempOutputPath);
  } catch (error) {
    console.error("Error downloading thumbnail:", err
  }
}
```

This modification downloads the thumbnail to a temporary file, resizes it to fit Plex's aspect ratio, and then saves the resized image to the output path.

Now, when you run the script, it should download and resize the thumbnails to fit properly in Plex.



```
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker$
node playlist_download.mjs "https://www.youtube.com/playlist?
list=PLFm1tTY1NA4ebglc7QWni3Vx6Zzdx0PQu" Hermitcraft-9
comparing with previous downloads in:
downloads/downloaded_videos_PLFm1tTY1NA4ebglc7QWni3Vx6Zzdx0PQu.json
```

Please write in English language.

Here's the modified `downloadVideo` function:

And the loop where you call this function should be updated to:

```
    episodeCounter++;  
  }  
}
```

Now, the script should work as expected.



Is there some way without too many lines of code to show download progress? It's fine if not.

Please write in English language.

Yes, you can display download progress with a simple progress bar. You can modify the `downloadVideo` function to include a progress bar using the `response.headers` to get the content length and update the progress based on the data received.



Here's the updated `downloadVideo` function with a progress bar:

javascript

Copy code

```
async function downloadVideo(videoInfo, filename) {  
  return new Promise(async (resolve, reject) => {  
    try {  
      const response = await fetch(videoInfo.url);  
      const totalBytes = parseInt(response.headers.get('content-length'));  
      let downloadedBytes = 0;  
  
      const progressBarLength = 30;  
  
      const updateProgressBar = () => {  
        const progressPercentage = (downloadedBytes / totalBytes) * 100;  
        const progressBarFilled = Math.floor(progressPercentage * progressBarLength);  
        const progressBarEmpty = progressBarLength - progressBarFilled;  
        const progressBar = '[' + '='.repeat(progressBarFilled) + ' '.repeat(progressBarEmpty) + ']' +  
          `${progressPercentage}%`;   
  
        process.stdout.clearLine();  
        process.stdout.cursorTo(0);  
        process.stdout.write(`Downloading: ${progressBar}`);  
      };  
  
      const fileStream = fs.createWriteStream(filename);  
      response.body.on('data', (chunk) => {  
        downloadedBytes += chunk.length;  
        updateProgressBar();  
      });  
  
      await pipeline(response.body, fileStream);  
  
      const thumbnailFilename = `${videoInfo.id}.jpg`  
      const thumbnailOutputPath = path.join(outputDir, thumbnailFilename);  
  
      await downloadThumbnail(videoInfo.thumbnail, thumbnailOutputPath);  
  
      console.log(); // Print a newline after the progress bar  
      resolve();  
    } catch (error) {  
      fs.unlink(filename, () => {});  
      reject(error);  
    }  
  });  
}
```

This progress bar will update as the video downloads, showing the percentage of completion. Note that this solution is for Node.js environments and may not work correctly in some terminals.



```
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker$  
node playlist_download.mjs "https://www.youtube.com/playlist?list=PLFm1tTY1NA4ebglc7QWni3Vx6Zzdx0PQu" Hermitcraft-9  
comparing with previous downloads in:  
downloads/downloaded_videos_PLFm1tTY1NA4ebglc7QWni3Vx6Zzdx0PQu.json
```

Getting info for videold: kXV608\_KsHA kXV608\_KsHA, \_AD7tSwoWX4,  
KYOspUantFM, HpcMKZUs2QY, ItCKu\_WlejA, pxE9jTx9BK4, 1WwGneY9rhQ, A-  
LpG6siR5E, T4D-D\_eDLeA, rTfxjtSaBPY, LQ6iO-FF53l, ymRPv8AsodA  
Getting info for videold: kXV608\_KsHA  
Downloading video: Hermitcraft 9: Episode 1 - RICHEST HERMIT  
Error: TypeError [ERR\_INVALID\_URL]: Invalid URL

Please write in English language.

Apologies for the oversight. It appears that the issue is caused by the ``videoInfo.url`` not being recognized as a valid URL. Instead of passing the whole ``videoInfo`` object to the ``downloadVideo`` function, you should pass only the ``videoUrl`` and the ``thumbnail`` URL as separate arguments.

Here's the updated section of the code where you call ``downloadVideo``:

```
javascript Copy code

for (const videoId of videoIds) {
  console.log(`Getting info for videoId: ${videoId}`)
  const videoInfo = await getVideoInfo(videoId);
  const title = videoInfo.title;
  const videoUrl = videoInfo.url;
  const thumbnailUrl = videoInfo.thumbnail;

  const fileName = `${outputDir}/${seasonCounter
    .toString()
    .padStart(2, "0")}${episodeCounter.toString().pa
    /[\/:*?"<>|]/g,
    " "
  }).mp4`;

  console.log(`Downloading video: ${title}`);
  await downloadVideo(videoUrl.trim(), thumbnailUrl,
  console.log(`Downloaded video: ${title}`);
  ...
}
```

And modify the ``downloadVideo`` function definition like this:

```
javascript Copy code

async function downloadVideo(videoUrl, thumbnailUrl,
  ...
}
```

This change should resolve the error. If you still encounter issues, please let me know.



Instead I just added videoUrl to videoInfo. Here's the new loop code:

```
for (const videold of videolds) {
  console.log(` Getting info for videold: ${videold}`);
  const { stdout: videoUrl } = await execPromisified(
    `yt-dlp -i -f best --cookies ${cookiesFilePath} --get-url
https://www.youtube.com/watch?v=${videold}`,
    { maxBuffer: 10 * 1024 * 1024 }
  );

  const videoInfo = await getVideoInfo(videold);
  videoInfo.url = videoUrl;
  const title = videoInfo.title;
  const videoUploadDate = videoInfo.upload_date;

  const currentYear = parseInt(videoUploadDate.slice(0, 4),
10);
  const firstYear = parseInt(firstVideoUploadDate.slice(0, 4),
10);
  const seasonCounter = currentYear - firstYear + 1;

  const fileName = `${outputDir}/${seasonCounter
```

```

        .toString()
        .padStart(2,
"0"))E${episodeCounter.toString().padStart(2,
"0"))-${title.replace(
/[/:.*?*<>|]/g,
" ")
}).mp4`;

console.log(` Downloading video: ${title}`);
await downloadVideo(videoInfo, fileName); // pass the
videoInfo object
console.log(` Downloaded video: ${title}`);

// Append downloaded video data to the file
downloadedData.push({
  id: videoId,
  season: seasonCounter,
  episode: episodeCounter,
  url: videoUrl.trim(),
});
await writeFile(downloadedVideosFile,
JSON.stringify(downloadedData, null, 2));

episodeCounter++;
}

```

Please write in English language.

Alright, that should work as well. By adding the `videoUrl` to the `videoInfo` object, you can continue using the `videoInfo` object in the `downloadVideo` function. The updated loop code you provided should not encounter the "Invalid URL" error anymore.

If you have any other issues or need further assistance, please feel free to ask.



It looks like the thumbnails aren't getting resized to the proper aspect ratio

Please write in English language.

I apologize for the confusion earlier. Here's a modified version of the `downloadThumbnail` function that resizes the thumbnail to the correct aspect ratio for Plex (16:9). We will be using the `sharp` package for image resizing, so you will need to install it:

```

bash
Copy code

npm install sharp

```

Then, you can modify the `downloadThumbnail` function like this:

```

javascript
Copy code

import sharp from "sharp";

async function downloadThumbnail(thumbnailUrl, output
try {
  const response = await fetch(thumbnailUrl);
  const buffer = await response.buffer();

  const resizedImageBuffer = await sharp(buffer)
    .resize({ width: 640, height: 360, fit: "cover"
    .toBuffer();

  await fs.promises.writeFile(outputPath, resizedIm
} catch (error) {
  console.error("Error downloading thumbnail:", err
}
}

```

This updated function will:

1. Fetch the thumbnail from the given URL and store it in a buffer.
2. Use `sharp` to resize the image to a 16:9 aspect ratio with dimensions of 640×360 pixels (you can adjust the width and height as needed).
3. Save the resized thumbnail to the specified `outputPath`.

After updating the `downloadThumbnail` function, the thumbnails should be resized to the correct aspect ratio for Plex.



DO we actually even need to resize the thumbnails? What is the difference between youtube and plex for video thumbnail sizes?

Please write in English language.

There might be no strict need to resize the thumbnails, but it could improve the visual appearance of the thumbnails in Plex. YouTube and Plex may display thumbnails with different aspect ratios or sizes, which could affect how they appear in the respective interfaces.



YouTube usually displays video thumbnails at a 16:9 aspect ratio. However, the actual dimensions can vary depending on the device or interface where they are displayed. For example, in search results or on the homepage, the dimensions can be 336×188 pixels, while in other places they might be displayed at different sizes.

Plex also prefers a 16:9 aspect ratio for thumbnails. The difference in appearance between YouTube and Plex might arise from the different interfaces, thumbnail sizes, or other styling factors. Resizing the thumbnail to a specific size ensures that it is displayed consistently in Plex, regardless of the original dimensions of the YouTube thumbnail.

If you find that the thumbnails look good enough without resizing, you can skip the resizing step. However, if you want to ensure a consistent appearance across all thumbnails in Plex, resizing them to a standard size with a 16:9 aspect ratio is recommended.



I'm going to just delete the thumbnail resize code. I don't think I need it. I do need a way to generate posters though. Is there some way to use the playlist title to generate posters for the seasons by having an image generated by AI and then overlaying the image with the name of the playlist and season?

Please write in English language.

Yes, you can generate season posters using AI-generated images and overlay them with the playlist title and season number. Here's a high-level outline of how you can achieve this:

1. Use an AI-generated image API to get a relevant image: There are various AI-generated image services available, such as DALL-E or RunwayML, which can generate images based on a given text prompt. You can use the playlist title as the prompt to generate a relevant image.
2. Overlay text on the image: To overlay the playlist title and season number on the image, you can use a library like `sharp` in Node.js.

Here's an example of how you can overlay text on an image using the `sharp` library:

javascript

Copy code

```

import sharp from 'sharp';
import { readFile, writeFile } from 'fs/promises';

async function generatePoster(inputImage, outputImage) {
  const image = sharp(inputImage);
  const metadata = await image.metadata();

  const fontSize = Math.round(metadata.width / 10);
  const text = `Season ${season}: ${title}`;

  await image
    .composite([
      {
        input: Buffer.from(`<svg xmlns="http://www.w3
          <text x="20" y="${fontSize + 20}" font-size
            ${text}
          </text>
          </svg>`),
      },
    ])
    .toFile(outputImage);
}

(async () => {
  const inputImage = await readFile('input.jpg');
  const title = 'Your Playlist Title';
  const season = 1;
  const outputImage = 'output.jpg';

  await generatePoster(inputImage, outputImage, title
})();

```

Replace 'input.jpg' with the path to your AI-generated image, and modify 'title' and 'season' variables as needed.

This code reads an input image, generates an SVG text overlay with the specified title and season, and then composites the text onto the image using `sharp`. Finally, the resulting image is saved as 'output.jpg'.

You can adapt this code to your specific use case, downloading an AI-generated image based on the playlist title and then overlaying the text as shown above.



In the following code, it looks like seasonCounter is getting declared twice (once before the loop and again inside the loop). Before the loop it is correctly checking for previously downloaded videos and continuing the seasons from there, but that seasonCounter doesn't appear to be getting used and instead the seasons are perhaps being miscalculated inside of the loop. Is this code correct, or do you need to change it?

```

(async () => {
  try {
    const { stdout: ids } = await execPromisified(
      `yt-dlp -i -f best --cookies ${cookiesFilePath} --get-id
        ${playlistUrl}`
    );

    let downloadedData = [];
    if (fs.existsSync(downloadedVideosFile)) {
      const content = await readFile(downloadedVideosFile, "utf-8");
      downloadedData = JSON.parse(content);
    }

    const videolds = ids
      .split("\n")
      .filter((id) => id && !downloadedData.some((entry) =>
        entry.id === id));

    if (playlistUrl.includes("videos")) {
      videolds.reverse();
    }

    console.log(`Getting info for videold: ${videolds[0]}`,
      videolds.join(", "));
  }
}

```



```

const firstVideoInfo = await getVideoInfo(videoIds[0]);
const firstVideoUploadDate = firstVideoInfo.upload_date;

let seasonCounter;
let episodeCounter;

if (downloadedData.length > 0) {
  const lastEntry = downloadedData[downloadedData.length
- 1];
  seasonCounter = lastEntry.season;
  episodeCounter = lastEntry.episode + 1;
} else {
  const firstYear = parseInt(firstVideoUploadDate.slice(0, 4),
10);
  seasonCounter = 1;
  episodeCounter = 1;
}

for (const videoId of videoIds) {
  console.log(` Getting info for videoId: ${videoId} `);
  const { stdout: videoUrl } = await execPromisified(
    `yt-dlp -i -f best --cookies ${cookiesFilePath} --get-url
https://www.youtube.com/watch?v=${videoId} `,
    { maxBuffer: 10 * 1024 * 1024 }
  );

  const videoInfo = await getVideoInfo(videoId);
  videoInfo.url = videoUrl;
  const title = videoInfo.title;
  const videoUploadDate = videoInfo.upload_date;

  const currentYear = parseInt(videoUploadDate.slice(0, 4),
10);
  const firstYear = parseInt(firstVideoUploadDate.slice(0, 4),
10);
  const seasonCounter = currentYear - firstYear + 1;

  const fileName = `${outputDir}/${S}${seasonCounter
.toString()
.padStart(2,
"0")}${episodeCounter.toString().padStart(2,
"0")}-${title.replace(
/[/:.*?*<>|]/g,
" "
)}.mp4`;

  console.log(` Downloading video: ${title} `);
  await downloadVideo(videoInfo, fileName); // pass the
videoInfo object
  console.log(` Downloaded video: ${title} `);

  // Append downloaded video data to the file
  downloadedData.push({
    id: videoId,
    season: seasonCounter,
    episode: episodeCounter,
    url: videoUrl.trim(),
  });
  await writeFile(downloadedVideosFile,
JSON.stringify(downloadedData, null, 2));

  episodeCounter++;
}
} catch (error) {
  console.error(` Error: ${error} `);
}
})();

```

Please write in English language.

You are correct; there is an issue in the code where the `seasonCounter` is declared twice. You should remove the second declaration inside the loop and use the previously declared `seasonCounter` variable to correctly calculate the seasons. Here is the corrected code:

```

(async () => {
  try {
    // ... (previous code remains the same)

    for (const videoId of videoIds) {
      console.log('Getting info for videoId: ${videoId}')
      // ... (previous code remains the same)

      const currentYear = parseInt(videoUploadDate.slice(0, 4))
      const firstYear = parseInt(firstVideoUploadDate.slice(0, 4))
      seasonCounter = currentYear - firstYear + 1; //

      // ... (rest of the code remains the same)
    }
  } catch (error) {
    console.error('Error: ${error}');
  }
})();

```

By removing the second declaration of `seasonCounter`, the code will now correctly use the previously declared variable and calculate the seasons based on previously downloaded videos.



But then what is this code even doing? It doesn't look like it gets used:

```

let seasonCounter;
let episodeCounter;

if (downloadedData.length > 0) {
  const lastEntry = downloadedData[downloadedData.length - 1];
  seasonCounter = lastEntry.season;
  episodeCounter = lastEntry.episode + 1;
} else {
  const firstYear = parseInt(firstVideoUploadDate.slice(0, 4), 10);
  seasonCounter = 1;
  episodeCounter = 1;
}

```

Please write in English language.

This code initializes the `seasonCounter` and `episodeCounter` variables based on the previously downloaded videos, if any. It checks if there are any downloaded videos stored in `downloadedData`. If there are, it retrieves the season and episode number of the last downloaded video and continues the count from there. If no previous videos are found, it initializes both the season and episode counters to 1.



Here's a breakdown of what the code does:

1. Declare `seasonCounter` and `episodeCounter` variables.
2. Check if there are any previously downloaded videos in `downloadedData`.
3. If there are previous videos, set `seasonCounter` to the season number of the last downloaded video and set `episodeCounter` to the episode number of the last downloaded video plus 1 (to continue counting from the last episode).
4. If there are no previous videos, set both `seasonCounter` and `episodeCounter` to 1.

In the corrected code from my previous response, the `seasonCounter` variable is updated inside the loop based on the current video's upload year and the first video's upload year. This way, it should correctly calculate the season number based on the video upload year while still utilizing the initial values calculated before the loop.



No, the seasonCounter variable is never used between here

```
if (downloadedData.length > 0) {
  const lastEntry = downloadedData[downloadedData.length
- 1];
  seasonCounter = lastEntry.season;
  episodeCounter = lastEntry.episode + 1;
} else {
  const firstYear = parseInt(firstVideoUploadDate.slice(0, 4),
10);
  seasonCounter = 1;
  episodeCounter = 1;
}
```

and where you are setting it again here:

```
seasonCounter = currentYear - firstYear + 1;
```

Please write in English language.

You are correct. I apologize for the confusion in my previous responses. The initial `seasonCounter` variable is not being used between those two sections of code. To utilize the initial values and properly calculate the season number, you should remove the `const` keyword before the `seasonCounter` inside the loop, so it updates the existing variable rather than creating a new one.



Here's the corrected code:

```
javascript Copy code

for (const videoId of videoIds) {
  // ...
  const currentYear = parseInt(videoUploadDate.slic
const firstYear = parseInt(firstVideoUploadDate.s
seasonCounter = currentYear - firstYear + 1; // R

  // ...
}
```

This way, the code will use the initial value of `seasonCounter` and update it inside the loop based on the current video's upload year and the first video's upload year.



Wouldn't doing it this way work better?

```
(async () => {
  try {
    const { stdout: ids } = await execPromisified(
      `yt-dlp -i -f best --cookies ${cookiesFilePath} --get-id
${playlistUrl}`
    );

    let downloadedData = [];
    if (fs.existsSync(downloadedVideosFile)) {
      const content = await readFile(downloadedVideosFile, "utf-
8");
      downloadedData = JSON.parse(content);
    }

    const videoids = ids
      .split("\n")
      .filter((id) => id && !downloadedData.some((entry) =>
entry.id === id));

    if (playlistUrl.includes("videos")) {
      videoids.reverse();
    }
    let seasonCounter;
    let episodeCounter;
    let lastyear;

    if (downloadedData.length > 0) {
```

```

const lastEntry = downloadedData[downloadedData.length
- 1];
const lastEntryInfo = await getVideoInfo(lastEntry.id);
const lastEntryUploadDate = lastEntryInfo.upload_date;
lastyear = parseInt(lastEntryUploadDate.slice(0, 4), 10);
seasonCounter = lastEntry.season;
episodeCounter = lastEntry.episode + 1;
} else {
  console.log(` Getting info for videoid: ${videoids[0]} `,
videoids.join(", "));
const firstVideoInfo = await getVideoInfo(videoids[0]);
const firstVideoUploadDate = firstVideoInfo.upload_date;
lastyear = parseInt(firstVideoUploadDate.slice(0, 4), 10);
seasonCounter = 1;
episodeCounter = 1;
}

for (const videoid of videoids) {
  console.log(` Getting info for videoid: ${videoid} `);
const { stdout: videoUrl } = await execPromisified(
`yt-dlp -i -f best --cookies ${cookiesFilePath} --get-url
https://www.youtube.com/watch?v=${videoid} `,
{ maxBuffer: 10 * 1024 * 1024 }
);

const videoInfo = await getVideoInfo(videoid);
videoInfo.url = videoUrl;
const title = videoInfo.title;
const videoUploadDate = videoInfo.upload_date;

const currentYear = parseInt(videoUploadDate.slice(0, 4),
10);
if (currentYear > lastyear) {
  seasonCounter++;
  episodeCounter = 1;
}

const fileName = `${outputDir}/S${seasonCounter}
.toString()
.padStart(2,
"0"))E${episodeCounter.toString().padStart(2,
"0"))-${title.replace(
/[/:.*?*<>|]/g,
" "
)}.mp4`;

console.log(` Downloading video: ${title} `);
await downloadVideo(videoInfo, fileName); // pass the
videoInfo object
console.log(` Downloaded video: ${title} `);

// Append downloaded video data to the file
downloadedData.push({
  id: videoid,
  season: seasonCounter,
  episode: episodeCounter,
  url: videoUrl.trim(),
});
await writeFile(downloadedVideosFile,
JSON.stringify(downloadedData, null, 2));

  episodeCounter++;
}
} catch (error) {
  console.error(` Error: ${error} `);
}
})();

```

Please write in English language.

Yes, this approach would work better, as it takes into account the year of the last downloaded video, and updates the season and episode counter accordingly. This way, you can correctly continue from where you left off in case there are any new videos added to the playlist.

This code will correctly increment the season and reset the episode counter to 1 whenever a new year is encountered in the video upload dates. Good job on updating the code!



```
const inputImage = await readFile('input.jpg');
const title = 'Your Playlist Title';
const season = 1;
const outputImage = 'output.jpg';
```

Please write in English language.

It seems like you are trying to read an input image file and provide a title and season number for a playlist, and then define the name of the output image file. Here's a clarification of the code snippet you provided:



javascript

Copy code

```
// Read the input image file
const inputImage = await readFile('input.jpg');

// Set the title and season number for the playlist
const title = 'Your Playlist Title';
const season = 1;

// Define the output image file name
const
```



```
(async () => {
  try {
    const { stdout: ids } = await execPromisified(
      `yt-dlp -i -f best --cookies ${cookiesFilePath} --get-id ${playlistUrl}`
    );

    let downloadedData = [];
    if (fs.existsSync(downloadedVideosFile)) {
      const content = await readFile(downloadedVideosFile, "utf-8");
      downloadedData = JSON.parse(content);
    }

    const videoids = ids
      .split("\n")
      .filter((id) => id && !downloadedData.some((entry) => entry.id === id));

    if (playlistUrl.includes("videos")) {
      videoids.reverse();
    }

    let seasonCounter;
    let episodeCounter;
    let lastyear;

    if (downloadedData.length > 0) {
      const lastEntry = downloadedData[downloadedData.length - 1];
      const lastEntryInfo = await getVideoInfo(lastEntry.id);
      const lastEntryUploadDate = lastEntryInfo.upload_date;
      lastyear = parseInt(lastEntryUploadDate.slice(0, 4), 10);
      seasonCounter = lastEntry.season;
      episodeCounter = lastEntry.episode + 1;
    } else {
      console.log(`Getting info for videoid: ${videoids[0]}`, videoids.join(", "));
      const firstVideoInfo = await getVideoInfo(videoids[0]);
      const firstVideoUploadDate = firstVideoInfo.upload_date;
      lastyear = parseInt(firstVideoUploadDate.slice(0, 4), 10);
      seasonCounter = 1;
      episodeCounter = 1;
    }

    for (const videoid of videoids) {
      console.log(`Getting info for videoid: ${videoid}`);
      const { stdout: videoUrl } = await execPromisified(
        `yt-dlp -i -f best --cookies ${cookiesFilePath} --get-url https://www.youtube.com/watch?v=${videoid}`,
        { maxBuffer: 10 * 1024 * 1024 }
      );
    }
  }
})
```

```

    );

    const videoInfo = await getVideoInfo(videoId);
    videoInfo.url = videoUrl;
    const title = videoInfo.title;
    const videoUploadDate = videoInfo.upload_date;

    const currentYear = parseInt(videoUploadDate.slice(0, 4),
10);
    if (currentYear > lastyear) {
        seasonCounter = seasonCounter + 1;
        episodeCounter = 1;
        lastyear = currentYear;

        const inputImage = await readFile("input.jpg");
        // ChatGPT, please complete the following:
        // get the name of the playlist as it exists on youtube
        // correctly calculate the name of the output poster image
so plex can find it for the season
        // call generatePoster with the correct parameters
    }

    const fileName = `${outputDir}/${seasonCounter}
        .toString()
        .padStart(2,
"0"))E${episodeCounter.toString().padStart(2,
"0"))-${title.replace(
        /\[\/.*?*<>[]/g,
        " "
        )}.mp4`;

    console.log(` Downloading video: ${title}`);
    await downloadVideo(videoInfo, fileName);
    console.log(` Downloaded video: ${title}`);

    // Append downloaded video data to the file
    downloadedData.push({
        id: videoId,
        season: seasonCounter,
        episode: episodeCounter,
        url: videoUrl.trim(),
    });
    await writeFile(downloadedVideosFile,
JSON.stringify(downloadedData, null, 2));

    episodeCounter++;
}
} catch (error) {
    console.error(` Error: ${error}`);
}
})();

```

Please write in English language.

In your code, you need to retrieve the name of the playlist, calculate the output poster image's name, and call the `generatePoster` function. Here's an updated version of your code that does this:

```

javascript  Copy code

// ... previous code
const currentYear = parseInt(videoUploadDate.slice(0,
if (currentYear > lastyear) {
    seasonCounter = seasonCounter + 1;
    episodeCounter = 1;
    lastyear = currentYear;

    const inputImage = await readFile("input.jpg");

    // Get the name of the playlist as it exists on Y
    const playlistInfo = await getPlaylistInfo(playli
    const playlistTitle = playlistInfo.title;

    // Calculate the name of the output poster image
    const outputImage = `Season ${seasonCounter} Post

    // Call generatePoster with the correct parameter
    await generatePoster(inputImage, playlistTitle, s

```

```
}  
// ... rest of the code
```

This assumes you have a `getPlaylistInfo()` function to retrieve the playlist's information and a `generatePoster()` function to create the poster. You'll need to replace these function calls with the appropriate functions for your implementation.



Can you make a `getPlaylistInfo` function for me? It should be able to handle actual playlists and normal video listing pages for youtube channels

Please write in English language.

Sure! Here's a `getPlaylistInfo` function that uses `yt-dlp` to get information about a playlist or a channel's video listing page. This function should be added outside your main `(async () => {})` block:

```
javascript Copy code  
  
const { promisify } = require("util");  
const exec = promisify(require("child_process").exec)  
  
async function getPlaylistInfo(url) {  
  try {  
    const { stdout: jsonInfo } = await exec(  
      `yt-dlp --cookies ${cookiesFilePath} --sk`  
    );  
  
    const parsedInfo = JSON.parse(jsonInfo);  
    const playlistInfo = {  
      id: parsedInfo.playlist_id || parsedInfo.  
      title: parsedInfo.playlist_title || parse  
    };  
  
    return playlistInfo;  
  } catch (error) {  
    console.error('Error getting playlist info: $  
    return null;  
  }  
}
```

This function uses `yt-dlp` to retrieve JSON-formatted information about the provided URL. It then extracts the relevant data (playlist/channel ID and title) and returns an object with that information.

You can use this function in your main `(async () => {})` block to retrieve the playlist or channel information as needed.



```
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker$  
node playlist_download.mjs "https://www.youtube.com/playlist?  
list=PLFm1tTY1NA4ebglc7QWni3Vx6Zzdx0PQu" Hermitcraft-9  
comparing with previous downloads in:  
downloads/downloaded_videos_PLFm1tTY1NA4ebglc7QWni3Vx6Zzdx0PQu.json  
Getting info for videoid: kXV608_KsHA kXV608_KsHA, _AD7tSwoWX4,  
KYOsPUantFM, HpcMKZUs2QY, ItCKu_WlejA, pxE9jTx9BK4, 1WwGneY9rhQ, A-  
LpG6siR5E, T4D-D_eDLeA, rTfxjtSaBPY, LQ6iO-FF53l, ymRPv8AsodA  
Getting info for videoid: kXV608_KsHA  
Downloading video: Hermitcraft 9: Episode 1 - RICHEST HERMIT  
Downloading: [=====] 100%  
Downloaded video: Hermitcraft 9: Episode 1 - RICHEST HERMIT  
Getting info for videoid: _AD7tSwoWX4  
Downloading video: Hermitcraft 9: Episode 2 - WEIRD GIANT FARM  
Downloading: [=====] 100%  
Downloaded video: Hermitcraft 9: Episode 2 - WEIRD GIANT FARM  
Getting info for videoid: KYOsPUantFM  
Downloading video: Hermitcraft 9: Episode 3 - A DANGEROUS HOLE  
Downloading: [=====] 100%  
Downloaded video: Hermitcraft 9: Episode 3 - A DANGEROUS HOLE
```


```
Getting info for videoid: HpcMKZUs2QY
Downloading video: Hermitcraft 9: Episode 4 - MEGA BASE PLANS
Downloading: [=====] 100%
Downloaded video: Hermitcraft 9: Episode 4 - MEGA BASE PLANS
Getting info for videoid: ItCKu_WlejA
Downloading video: Hermitcraft 9: Episode 5 - FIRST DEFENCES
Downloading: [=====] 100%
Downloaded video: Hermitcraft 9: Episode 5 - FIRST DEFENCES
Getting info for videoid: pxE9JTx9BK4
Downloading video: Hermitcraft 9: Episode 6 - GRIAN PROOF VAULT
Downloading: [=====] 100%
Downloaded video: Hermitcraft 9: Episode 6 - GRIAN PROOF VAULT
Getting info for videoid: 1WwGneY9rhQ
Error getting playlist info: SyntaxError: Unexpected token o in JSON at position 1
Error: TypeError: Cannot read properties of null (reading 'title')
```

Please write in English language.

It seems like the ``yt-dlp`` command in the ``getPlaylistInfo`` function is returning output that cannot be parsed as JSON, causing a ``SyntaxError``. To handle this, we can add a check for valid JSON before parsing it.

Additionally, it seems like an error occurred while fetching video info for videoid ``1WwGneY9rhQ``. To handle errors during the execution of the main loop, we can wrap the inside of the loop in a try-catch block.


Here's an updated version of your main ``(async () => {})`` block with the suggested changes:

```
javascript  Copy code

(async () => {
  try {
    // ... (previous code before the loop)

    for (const videoId of videoIds) {
      try {
        // ... (previous code inside the loop)
      } catch (innerError) {
        console.error(`Error processing videoId ${videoId}: ${innerError}`);
      }
    }
  } catch (error) {
    console.error(`Error: ${error}`);
  }
})();
```

And here's the updated ``getPlaylistInfo`` function with a check for valid JSON:

```
javascript  Copy code

async function getPlaylistInfo(url) {
  try {
    const { stdout: jsonInfo } = await exec(
      `yt-dlp --cookies ${cookiesFilePath} --skip-dow`
    );

    if (!isValidJSON(jsonInfo)) {
      throw new Error("Invalid JSON received from yt-");
    }

    const parsedInfo = JSON.parse(jsonInfo);
    const playlistInfo = {
      id: parsedInfo.playlist_id || parsedInfo.channe
      title: parsedInfo.playlist_title || parsedInfo.
    };

    return playlistInfo;
  } catch (error) {
    console.error(`Error getting playlist info: ${err
    return null;
  }
}
```



```
function isValidJSON(jsonString) {
  try {
    JSON.parse(jsonString);
    return true;
  } catch (e) {
    return false;
  }
}
```

These changes should help you identify and handle errors during the main loop execution and when parsing JSON from ``yt-dlp``.



When I console.log out jsonInfo it shows the following:

```
jsonInfo <ref *1> Socket {
  connecting: false,
  _hadError: false,
  _parent: null,
  _host: null,
  _closeAfterHandlingError: false,
  _readableState: ReadableState {
    objectMode: false,
    highWaterMark: 16384,
    buffer: BufferList { head: null, tail: null, length: 0 },
    length: 0,
    pipes: [],
    flowing: true,
    ended: false,
    endEmitted: false,
    reading: true,
    constructed: true,
    sync: false,
    needReadable: true,
    emittedReadable: false,
    readableListening: false,
    resumeScheduled: true,
    errorEmitted: false,
    emitClose: false,
    autoDestroy: true,
    destroyed: false,
    errored: null,
    closed: false,
    closeEmitted: false,
    defaultEncoding: 'utf8',
    awaitDrainWriters: null,
    multiAwaitDrain: false,
    readingMore: false,
    dataEmitted: false,
    decoder: StringDecoder {
      encoding: 'utf8',
      [Symbol(kNativeDecoder)]: <Buffer 00 00 00 00 00 00 01>
    },
    encoding: 'utf8',
    [Symbol(kPaused)]: false
  },
  _events: [Object: null prototype] {
    end: [Function: onReadableStreamEnd],
    close: [Function (anonymous)],
    data: [Function: onChildStdout]
  },
  _eventsCount: 3,
  _maxListeners: undefined,
  _writableState: WritableState {
    objectMode: false,
    highWaterMark: 16384,
    finalCalled: false,
    needDrain: false,
    ending: false,
    ended: false,
    finished: false,
    destroyed: false,
    decodeStrings: false,
    defaultEncoding: 'utf8',
    length: 0,
    writing: false,
    corked: 0,
    sync: true,
    bufferProcessing: false,
```

```

onwrite: [Function: bound onwrite],
writecb: null,
writelen: 0,
afterWriteTickInfo: null,
buffered: [],
bufferedIndex: 0,
allBuffers: true,
allNoop: true,
pendingcb: 0,
constructed: true,
prefinished: false,
errorEmitted: false,
emitClose: false,
autoDestroy: true,
errored: null,
closed: false,
closeEmitted: false,
[Symbol(kOnFinished)]: []
},
allowHalfOpen: false,
_sockname: null,
_pendingData: null,
_pendingEncoding: "",
server: null,
_server: null,
[Symbol(async_id_symbol)]: 414,
[Symbol(kHandle)]: Pipe { reading: true,
[Symbol(owner_symbol)]: [Circular *] },
[Symbol(lastWriteQueueSize)]: 0,
[Symbol(timeout)]: null,
[Symbol(kBuffer)]: null,
[Symbol(kBufferCb)]: null,
[Symbol(kBufferGen)]: null,
[Symbol(kCapture)]: false,
[Symbol(kSetNoDelay)]: false,
[Symbol(kSetKeepAlive)]: false,
[Symbol(kSetKeepAliveInitialDelay)]: 0,
[Symbol(kBytesRead)]: 0,
[Symbol(kBytesWritten)]: 0
}
Error getting playlist info: SyntaxError: Unexpected token o in
JSON at position 1
Error: TypeError: Cannot read properties of null (reading 'title')

```

Please write in English language.

It seems like the issue is with the way `exec` is being used. Instead of using `exec`, we can use `execFile` from the `child_process` module, which is more suitable for running external commands and capturing the output.

First, you'll need to import `execFile`:

```


javascript  Copy code

import { execFile } from "child_process";

```

Then you can create a new function called `execFilePromisified` to use `execFile` with promises:

```

javascript  Copy code

function execFilePromisified(command, args) {
  return new Promise((resolve, reject) => {
    execFile(command, args, (error, stdout, stderr) =
      if (error) {
        reject(error);
      } else {
        resolve({ stdout, stderr });
      }
    });
});
}

```

Now, you can update the `getPlaylistInfo` function to use `execFilePromisified` instead of `exec`:

```
javascript Copy code

async function getPlaylistInfo(url) {
  try {
    const { stdout: jsonInfo } = await execFilePromisified(
      "--cookies",
      cookiesFilePath,
      "--skip-download",
      "--print-json",
      url,
    );

    if (!isValidJSON(jsonInfo)) {
      throw new Error("Invalid JSON received from yt-");
    }

    const parsedInfo = JSON.parse(jsonInfo);
    const playlistInfo = {
      id: parsedInfo.playlist_id || parsedInfo.channel_id,
      title: parsedInfo.playlist_title || parsedInfo.channel_title,
    };

    return playlistInfo;
  } catch (error) {
    console.error(`Error getting playlist info: ${error}`);
    return null;
  }
}
```

Now, when you run your script, it should correctly capture the output of `yt-dlp` and avoid the previous error.



```
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker$
node playlist_download.mjs "https://www.youtube.com/playlist?
list=PLFm1tTY1NA4ebglc7QWni3Vx6Zzdx0PQu" Hermitcraft-9
comparing with previous downloads in:
downloads/downloaded_videos_PLFm1tTY1NA4ebglc7QWni3Vx6Zzdx0PQu.json
Getting info for videoid: kXV608_KsHA kXV608_KsHA, _AD7tSwoWX4,
KYOsPUantFM, HpcMKZUs2QY, ItCKu_WlejA, pxE9JTx9BK4, 1WwGneY9rhQ, A-
LpG6siR5E, T4D-D_eDLeA, rTfxtSaBPY, LQ6IO-FF53I, ymRPv8AsodA
Error getting playlist info: RangeError
[ERR_CHILD_PROCESS_STDIO_MAXBUFFER]: stdout maxBuffer length
exceeded
Error: TypeError: Cannot read properties of null (reading 'title')
```

Please write in English language.

This error occurs when the output of `yt-dlp` exceeds the default max buffer size for `execFile`. To fix this, you can increase the max buffer size by passing an options object with a larger `maxBuffer` value to the `execFilePromisified` function.

Update the `execFilePromisified` function to accept an `options` parameter:

```
javascript Copy code

function execFilePromisified(command, args, options =
  return new Promise((resolve, reject) => {
    execFile(command, args, options, (error, stdout,
      if (error) {
        reject(error);
      } else {
        resolve({ stdout, stderr });
      }
    });
  });
}
```

Then, increase the `maxBuffer` value when calling `execFilePromisified` in the `getPlaylistInfo` function:

```
javascript Copy code

async function getPlaylistInfo(url) {
  try {
    const { stdout: jsonInfo } = await execFilePromisified(
      "--cookies",
      cookiesFilePath,
      "--skip-download",
      "--print-json",
      url,
      [], { maxBuffer: 10 * 1024 * 1024 }); // Increase

    if (!isValidJSON(jsonInfo)) {
      throw new Error("Invalid JSON received from yt-");
    }

    const parsedInfo = JSON.parse(jsonInfo);
    const playlistInfo = {
      id: parsedInfo.playlist_id || parsedInfo.channel_id,
      title: parsedInfo.playlist_title || parsedInfo.title,
    };

    return playlistInfo;
  } catch (error) {
    console.error(`Error getting playlist info: ${error}`);
    return null;
  }
}
```

Now, when you run your script, it should work correctly and not exceed the max buffer size.



From now on, always prefer arrow functions

Please write in English language.

Sure, I will use arrow functions when writing code for you. Here is the updated code with arrow functions:

Update the `execFilePromisified` function:

```
javascript Copy code

const execFilePromisified = (command, args, options = {}) => {
  return new Promise((resolve, reject) => {
    execFile(command, args, options, (error, stdout, stderr) => {
      if (error) {
        reject(error);
      } else {
        resolve({ stdout, stderr });
      }
    });
  });
};
```

And the `getPlaylistInfo` function:

```
javascript Copy code

const getPlaylistInfo = async (url) => {
  try {
    const { stdout: jsonInfo } = await execFilePromisified(
      "--cookies",
      cookiesFilePath,
      "--skip-download",
      "--print-json",
      url,
      [], { maxBuffer: 10 * 1024 * 1024 });

    if (!isValidJSON(jsonInfo)) {
      throw new Error("Invalid JSON received from yt-");
    }

    const parsedInfo = JSON.parse(jsonInfo);
    const playlistInfo = {
      id: parsedInfo.playlist_id || parsedInfo.channel_id,
      title: parsedInfo.playlist_title || parsedInfo.title,
    };

    return playlistInfo;
  } catch (error) {
    console.error(`Error getting playlist info: ${error}`);
    return null;
  }
}
```

```

const parsedInfo = JSON.parse(jsonInfo);
const playlistInfo = {
  id: parsedInfo.playlist_id || parsedInfo.channe
  title: parsedInfo.playlist_title || parsedInfo.
};

return playlistInfo;
} catch (error) {
  console.error('Error getting playlist info: ${err
  return null;
}
};

```

Now the code uses arrow functions instead of traditional function declarations.



```

cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker$
node playlist_download.mjs "https://www.youtube.com/playlist?
list=PLFm1tTY1NA4ebglc7QWni3Vx6Zzdx0PQu" Hermitcraft-9
comparing with previous downloads in:
downloads/downloaded_videos_PLFm1tTY1NA4ebglc7QWni3Vx6Zzdx0PQu.json
Getting info for videoid: kXV608_KsHA kXV608_KsHA, _AD7tSwoWX4,
KYOsPUantFM, HpcMKZUs2QY, ItCKu_WlejA, pxE9jTx9BK4, 1WwGneY9rhQ, A-
LpG6siR5E, T4D-D_eDLeA, rTfxjtSaBPY, LQ6iO-FF53l, ymRPv8AsodA
Error getting playlist info: Error: Invalid JSON received from yt-dlp
Error: TypeError: Cannot read properties of null (reading 'title')

```

Please write in English language.

It seems that the `yt-dlp` command is still not returning valid JSON. Let's add more error handling and logging to understand what is happening.



Update the `getPlaylistInfo` function with additional error handling and logging:

```

javascript Copy code

const getPlaylistInfo = async (url) => {
  try {
    const { stdout: jsonInfo } = await execFilePromis
      "--cookies",
      cookiesFilePath,
      "--skip-download",
      "--print-json",
      url,
    ], { maxBuffer: 10 * 1024 * 1024 });

    if (!isValidJSON(jsonInfo)) {
      console.error('Invalid JSON received from yt-dl
      throw new Error("Invalid JSON received from yt-
    }

    const parsedInfo = JSON.parse(jsonInfo);
    const playlistInfo = {
      id: parsedInfo.playlist_id || parsedInfo.channe
      title: parsedInfo.playlist_title || parsedInfo.
    };

    return playlistInfo;
  } catch (error) {
    console.error('Error getting playlist info: ${err
    return null;
  }
};

```

Now, when the `yt-dlp` command does not return valid JSON, the console will log the output. This should help diagnose the issue. Run the script again and check the logged output if the error persists.



The output is way too long to paste here. It starts like this:

```
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker$ node playlist_download.mjs
"https://www.youtube.com/playlist?list=PLFm1tTY1NA4ebglc7QWni3Vx6Zzdx0PQu" Hermitcraft-9
comparing with previous downloads in:
downloads/downloaded_videos_PLFm1tTY1NA4ebglc7QWni3Vx6Zzdx0PQu.json
Getting info for videoid: kXV608_KsHA kXV608_KsHA, _AD7tSwoWX4, KYOsPUantFM, HpcMKZUs2QY,
ltCKu_WlejA, pxE9jTx9BK4, 1WwGneY9rhQ, A-LpG6siR5E, T4D-D_eDLeA, rTfxjtSaBPY, LQ6iO-FF53l,
ymRPv8AsodA
Invalid JSON received from yt-dlp:
{"id": "kXV608_KsHA", "title": "Hermitcraft 9: Episode 1 - RICHEST HERMIT", "formats": [{"format_id": "sb2",
"format_note": "storyboard", "ext": "mhtml", "protocol": "mhtml", "acodec": "none", "vcodec": "none", "url":
"https://i.ytimg.com/sb/kXV608_KsHA/storyboard3_L0/default.jpg?sqp=-
oaymwENSDFyq4qpAwVwAcABBqLzl_8DBgjB6pORBg==&sig=rs$AOn4CLAhZrb8d3wblxDKAN_TxDKvKxDtqw",
"width": 48, "height": 27, "fps": 0.07942811755361398, "rows": 10, "columns": 10, "fragments": [{"url":
"https://i.ytimg.com/sb/kXV608_KsHA/storyboard3_L0/default.jpg?sqp=-
oaymwENSDFyq4qpAwVwAcABBqLzl_8DBgjB6pORBg==&sig=rs$AOn4CLAhZrb8d3wblxDKAN_TxDKvKxDtqw",
"duration": 1259.0}], "resolution": "48x27", "aspect_ratio": 1.78, "http_headers": {"User-Agent": "Mozilla/5.0 (Windows
NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.20 Safari/537.36", "Accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8", "Accept-Language": "en-us,en;q=0.5", "Sec-
Fetch-Mode": "navigate"}, "audio_ext": "none", "video_ext": "none", "format": "sb2 - 48x27 (storyboard)"},
{"format_id": "sb1", "format_note": "storyboard", "ext": "mhtml", "protocol": "mhtml", "acodec": "none", "vcodec":
"none", "url": "https://i.ytimg.com/sb/kXV608_KsHA/storyboard3_L1/M$M.jpg?sqp=-
oaymwENSDFyq4qpAwVwAcABBqLzl_8DBgjB6pORBg==&sig=rs$AOn4CLDxcRicWSZqEF6bIG4gfZnligYkxw",
"width": 80, "height": 45, "fps": 0.10087370929308975, "rows": 10, "columns": 10, "fragments": [{"url":
"https://i.ytimg.com/sb

```

Please write in English language.

Based on the output you provided, it appears that the JSON string being returned by `yt-dlp` is actually valid JSON. The `isValidJSON` function might be returning a false negative. Let's update the `getPlaylistInfo` function to bypass the `isValidJSON` function and directly parse the JSON string to see if it works:



```
javascript Copy code

const getPlaylistInfo = async (url) => {
  try {
    const { stdout: jsonInfo } = await execFilePromis
      "--cookies",
      cookiesFilePath,
      "--skip-download",
      "--print-json",
      url,
      [], { maxBuffer: 10 * 1024 * 1024 });

    // Commenting out the following lines as they mig
    // if (!isValidJSON(jsonInfo)) {
    //   console.error('Invalid JSON received from yt
    //   throw new Error("Invalid JSON received from
    // }

    const parsedInfo = JSON.parse(jsonInfo);
    const playlistInfo = {
      id: parsedInfo.playlist_id || parsedInfo.channe
      title: parsedInfo.playlist_title || parsedInfo.
    };

    return playlistInfo;
  } catch (error) {
    console.error('Error getting playlist info: ${err
    return null;
  }
};

```

Run the script again and see if it works. If there are any errors, please provide the output.






It looks like the JSON may not actually be valid. I'm seeing this in the JSON:

```
&lsparams=mh%2Cmm%2Cmn%2Cms%2Cmv%2Cmvi%2Cpl%2Cinitcwndbps&sig=AG3C_xAwRglhAJVpdfuvoK5DAJ-Y7eRjjsnp2yVFPc
AMXOPrOLPbsQf0Oyg0O7EcxlN2Or5oHHbWdiOFv2ryaz&itag=251&source=youtube&requiresl=yes&mh=nq&mm=31%2C29&mn=sn-vg
AUTfPt1tD7ZdMin&vprv=1&svpuc=1&mime=audio%2Fwebm&gir=yes&clen=17226096&dur=1090.921&imt=1679591202438343&mt=168;

```

-27BETdPRKfoaJUINGU1L8PKsYnE%3D&isparams=mh%2Cmm%2Cmn%2Cms%2Cmv%2Cmvi%2Cpl%2Cinitcwndbps&isig=AG3C\_xAwRg  
{"id": "LQ6IO-FF53I", "title": "Hermitcraft 9: Episode 11 - MEGA BASE BEGINS", "formats": [{"format\_id": "sb2", "format\_note": "storyboard", "ex  
"fragments": [{"url": "https://i.ytimg.com/sb/LQ6IO-FF53I/storyboard3\_L0/default.jpg?sqp=-oaymwENSDFyq4qpAwVwAcABBqLzL\_8DBgia  
"text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8", "Accept-Language": "en-us,en;q=0.5", "Sec-Fetch-Mode": "navigate"}, "a  
oaymwENSDFyq4qpAwVwAcABBqLzL\_8DBgiaMahBg==&sig=rs\$AO4CLDM-7K

I think there is a new different JSON object being downloaded for each video in the playlist. That's not what we need though, we just need to  
Please write in English language.

I see what's happening now. You're right; the script is trying to     
parse multiple JSON objects as a single JSON object. You need to  
update the script to handle each JSON object separately. Modify  
the `getPlaylistInfo` function as shown below:

javascript  Copy code

```
const getPlaylistInfo = async (url) => {  
  try {  
    const { stdout: jsonInfo } = await execFilePromis  
      "--cookies",  
      cookiesFilePath,  
      "--skip-download",  
      "--print-json",  
      url,  
      ], { maxBuffer: 10 * 1024 * 1024 });  
  
    const jsonObjects = jsonInfo.split('\n').filter(o  
    let playlistInfo = null;  
  
    for (const jsonObject of jsonObjects) {  
      try {  
        const parsedInfo = JSON.parse(jsonObject);  
  
        playlistInfo = {  
          id: parsedInfo.playlist_id || parsedInfo.ch  
          title: parsedInfo.playlist_title || parsedI  
        };  
  
        if (playlistInfo.id && playlistInfo.title) {  
          break;  
        }  
      } catch (error) {  
        console.error(`Error parsing JSON object: ${e  
      }  
    }  
  
    if (!playlistInfo || !playlistInfo.id || !playlis  
      throw new Error("Unable to get playlist or chan  
    }  
  
    return playlistInfo;  
  } catch (error) {  
    console.error(`Error getting playlist info: ${err  
    return null;  
  }  
};
```

This updated function splits the output into separate lines and  
tries to parse each line as a JSON object. It searches for the  
`'playlist_id'` and `'playlist_title'` or `'channel_id'` and  
`'channel'`, and if found, it breaks the loop and returns the  
information. If it cannot find the required information in any of the  
JSON objects, it throws an error.

Try running the script again with these changes and let me know  
if it works as expected.



I'm not getting an error anymore, but I don't see the poster file  
anywhere

Please write in English language.

I'm glad the error is resolved. Now, let's address the issue of the missing poster file. It seems like the script provided earlier does not include the functionality to download the poster file. To add this functionality, you can modify the `downloadVideo` function as shown below:



```
javascript Copy code

const downloadVideo = async (videoId, outputPath) =>
  try {
    const { stdout: jsonInfo } = await execFilePromis
      "--cookies",
      cookiesFilePath,
      "--skip-download",
      "--print-json",
      `https://www.youtube.com/watch?v=${videoId}`,
    ];

    const videoInfo = JSON.parse(jsonInfo);

    const bestThumbnail = videoInfo.thumbnails.reduce
      return prev.height > current.height ? prev : cu
    );

    const thumbnailUrl = bestThumbnail.url;
    const posterFileName = path.join(outputPath, `${v

    await downloadFile(thumbnailUrl, posterFileName);

    console.log(`Downloaded poster for ${videoInfo.ti
  } catch (error) {
    console.error(`Error downloading video poster: ${
  }
};
```

Additionally, you'll need to create the `downloadFile` function, which downloads a file from a given URL and saves it to a specified path:

```
javascript Copy code

const downloadFile = async (url, outputPath) => {
  return new Promise((resolve, reject) => {
    const file = fs.createWriteStream(outputPath);
    https.get(url, (response) => {
      response.pipe(file);
      file.on("finish", () => {
        file.close(resolve);
      });
    }).on("error", (error) => {
      fs.unlink(outputPath, () => {});
      reject(error);
    });
  });
};
```

Don't forget to import the required modules at the beginning of your script:

```
javascript Copy code

const https = require("https");
const path = require("path");
```

After making these changes, the script should download the best-quality thumbnail for each video as a poster file and save it to the specified output directory. Run the script again and let me know if it works as expected.



No, I'll provide the current code. The generatePoster method doesn't seem to be saving a file anywhere (I can't find it). I run this code like this:  
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-



```
maker$ node playlist_download.mjs
"https://www.youtube.com/playlist?
list=PLFm1tTY1NA4ebglc7QWni3Vx6Zzdx0PQu" Hermitcraft-9
```

Where should the poster.jpg file be going?

```
import { exec, execFile } from "child_process";
import util from "util";
import fs from "fs";
import path from "path";
import axios from "axios";
const execPromisified = util.promisify(exec);
const writeFile = util.promisify(fs.writeFile);
const readFile = util.promisify(fs.readFile);
import { pipeline } from "stream/promises";
import fetch from "node-fetch";
import sharp from "sharp";

const playlistUrl = process.argv[2];
const folderName = process.argv[3];
const cookiesFilePath = "cookies.txt";

if (!fs.existsSync("downloads")) {
  fs.mkdirSync("downloads");
}

const execFilePromisified = (command, args, options = {}) => {
  return new Promise((resolve, reject) => {
    execFile(command, args, options, (error, stdout, stderr) => {
      if (error) {
        reject(error);
      } else {
        resolve({ stdout, stderr });
      }
    });
  });
};

const isValidJSON = (jsonString) => {
  try {
    JSON.parse(jsonString);
    return true;
  } catch (e) {
    return false;
  }
};

let downloadedVideosFile;
let outputDir;

if (playlistUrl.includes("playlist")) {
  const playlistId = new URL(playlistUrl).searchParams.get("list");
  downloadedVideosFile =
    `downloads/downloaded_videos_${playlistId}.json`;
  outputDir = ` /volume1/McCullohShare/Plex/TV/${folderName ?
folderName : playlistId}`;
} else if (playlistUrl.includes("videos")) {
  let channelName;

  try {
    ({ stdout: channelName } = await execPromisified(
      `yt-dlp --cookies ${cookiesFilePath} --get-filename --no-
playlist --output "%(uploader)s" "${playlistUrl}"`
    ));
    channelName = channelName.trim().split("\n")[0];
    console.log("channelName", channelName);
  } catch (error) {
    console.error(`Error getting channel name: ${error}`);
    process.exit(1);
  }

  downloadedVideosFile =
    `downloads/downloaded_videos_${channelName}.json`;
  outputDir = ` /volume1/McCullohShare/Plex/TV/${folderName ?
folderName : channelName}`;
} else {
  console.error("Invalid URL format. Please provide a playlist or
videos URL.");
  process.exit(1);
}
```

```

console.log("comparing with previous downloads in: ",
downloadedVideosFile);

if (fs.existsSync(outputDir)) {
  fs.mkdirSync(outputDir);
}

async function generatePoster(inputImage, outputImage, title,
season) {
  const image = sharp(inputImage);
  const metadata = await image.metadata();

  const fontSize = Math.round(metadata.width / 10);
  const text = `Season ${season}: ${title}`;

  await image
    .composite([
      {
        input: Buffer.from(`<svg
xmlns="http://www.w3.org/2000/svg" width="${
          metadata.width
        }" height="${metadata.height}">
<text x="20" y="${
          fontSize + 20
        }" font-size="${fontSize}" font-family="Arial" fill="white"
stroke="black" stroke-width="2">
${text}
</text>
</svg> `),
      },
    ])
    .toFile(outputImage);
}

async function downloadThumbnail(thumbnailUrl, outputPath) {
  try {
    const response = await axios({
      method: "GET",
      url: thumbnailUrl,
      responseType: "stream",
    });

    const tempOutputPath = `${outputPath}.jpg`;
    const writer = fs.createWriteStream(tempOutputPath);
    response.data.pipe(writer);

    await new Promise((resolve, reject) => {
      writer.on("finish", resolve);
      writer.on("error", reject);
    });

    fs.unlinkSync(tempOutputPath);
  } catch (error) {
    console.error("Error downloading thumbnail:", error.message);
  }
}

async function writeNfo(outputPath, videoInfo) {
  const nfoContent = `<?xml version="1.0" encoding="UTF-8"
standalone="yes" ?>
<episodedetails>
<title>${videoInfo.title}</title>
<plot>${videoInfo.description}</plot>
</episodedetails>
`;

  await writeFile(outputPath, nfoContent);
}

async function downloadVideo(videoInfo, filename) {
  return new Promise(async (resolve, reject) => {
    try {
      const response = await fetch(videoInfo.url);
      const totalBytes = parseInt(response.headers.get("content-
length"), 10);
      let downloadedBytes = 0;

      const progressBarLength = 30;

      const updateProgressBar = () => {
        const progressPercentage = (downloadedBytes /

```

```

totalBytes) * 100;
    const progressBarFilled = Math.floor(
        progressPercentage / (100 / progressBarLength)
    );
    const progressBarEmpty = progressBarLength -
progressBarFilled;
    const progressBar = ` ${"=".repeat(progressBarFilled)}${"
".repeat(
        progressBarEmpty
    )}`;

    process.stdout.clearLine();
    process.stdout.cursorTo(0);
    process.stdout.write(
        `Downloading: ${progressBar}
${Math.floor(progressPercentage)}% `
    );
    };
    };

    const fileStream = fs.createWriteStream(filename);
    response.body.on("data", (chunk) => {
        downloadedBytes += chunk.length;
        updateProgressBar();
    });

    await pipeline(response.body, fileStream);

    const thumbnailFilename = `${videoInfo.id}.jpg`;
    const thumbnailOutputPath = path.join(outputDir,
thumbnailFilename);

    await downloadThumbnail(videoInfo.thumbnail,
thumbnailOutputPath);

    const nfoFilename = `${path.basename(filename,
".mp4")}.nfo`;
    const nfoOutputPath = path.join(outputDir, nfoFilename);
    await writeNfo(nfoOutputPath, videoInfo);

    console.log(); // Print a newline after the progress bar
    resolve();
    } catch (error) {
        fs.unlink(filename, () => {});
        reject(error);
    }
    });
}

const getPlaylistInfo = async (url) => {
    try {
        const { stdout: jsonInfo } = await execFilePromisified(
            "yt-dlp",
            ["--cookies", cookiesFilePath, "--skip-download", "--print-json",
url],
            { maxBuffer: 10 * 1024 * 1024 }
        );

        const jsonObjects = jsonInfo.split("\n").filter((obj) => obj.trim()
!== "");
        let playlistInfo = null;

        for (const jsonObject of jsonObjects) {
            try {
                const parsedInfo = JSON.parse(jsonObject);

                playlistInfo = {
                    id: parsedInfo.playlist_id || parsedInfo.channel_id,
                    title: parsedInfo.playlist_title || parsedInfo.channel,
                };

                if (playlistInfo.id && playlistInfo.title) {
                    break;
                }
            } catch (error) {
                console.error(`Error parsing JSON object: ${error}`);
            }
        }

        if (!playlistInfo || !playlistInfo.id || !playlistInfo.title) {
            throw new Error("Unable to get playlist or channel
information");

```

```

    }

    return playlistInfo;
  } catch (error) {
    console.error(`Error getting playlist info: ${error}`);
    return null;
  }
};

async function getVideoInfo(videoId) {
  const { stdout: jsonOutput } = await execPromisified(
    `yt-dlp --cookies ${cookiesFilePath} -j https://www.youtube.com/watch?v=${videoId}`,
    { maxBuffer: 10 * 1024 * 1024 }
  );

  return JSON.parse(jsonOutput);
}

(async () => {
  try {
    const { stdout: ids } = await execPromisified(
      `yt-dlp -i -f best --cookies ${cookiesFilePath} --get-id ${playlistUrl}`
    );

    let downloadedData = [];
    if (fs.existsSync(downloadedVideosFile)) {
      const content = await readFile(downloadedVideosFile, "utf-8");
      downloadedData = JSON.parse(content);
    }

    const videoIds = ids
      .split("\n")
      .filter((id) => id && !downloadedData.some((entry) => entry.id === id));

    if (playlistUrl.includes("videos")) {
      videoIds.reverse();
    }

    let seasonCounter;
    let episodeCounter;
    let lastyear;

    if (downloadedData.length > 0) {
      const lastEntry = downloadedData[downloadedData.length - 1];
      const lastEntryInfo = await getVideoInfo(lastEntry.id);
      const lastEntryUploadDate = lastEntryInfo.upload_date;
      lastyear = parseInt(lastEntryUploadDate.slice(0, 4), 10);
      seasonCounter = lastEntry.season;
      episodeCounter = lastEntry.episode + 1;
    } else {
      console.log(`Getting info for videoId: ${videoIds[0]}`,
        videoIds.join(", "));
      const firstVideoInfo = await getVideoInfo(videoIds[0]);
      const firstVideoUploadDate = firstVideoInfo.upload_date;
      lastyear = parseInt(firstVideoUploadDate.slice(0, 4), 10);
      seasonCounter = 1;
      episodeCounter = 1;
    }

    const inputImage = await readFile("input.jpg");

    // Get the name of the playlist as it exists on YouTube
    const playlistInfo = await getPlaylistInfo(playlistUrl);
    const playlistTitle = playlistInfo.title;
    const outputImage = `poster.jpg`;

    await generatePoster(inputImage, playlistTitle, seasonCounter, outputImage);

    for (const videoId of videoIds) {
      console.log(`Getting info for videoId: ${videoId}`);
      const { stdout: videoUrl } = await execPromisified(
        `yt-dlp -i -f best --cookies ${cookiesFilePath} --get-url https://www.youtube.com/watch?v=${videoId}`,
        { maxBuffer: 10 * 1024 * 1024 }
      );

      const videoInfo = await getVideoInfo(videoId);

```

```

        videoInfo.url = videoUrl;
        const title = videoInfo.title;
        const videoUploadDate = videoInfo.upload_date;

        const currentYear = parseInt(videoUploadDate.slice(0, 4), 10);
        if (currentYear > lastyear) {
            seasonCounter = seasonCounter + 1;
            episodeCounter = 1;
            lastyear = currentYear;
        }

        const fileName = `${outputDir}/${S}${seasonCounter
            .toString()
            .padStart(2, "0")}${E}${episodeCounter.toString().padStart(2,
"0")}-${title.replace(
            /\[/.*?*<>[]/g,
            "-"
        )}.mp4`;

        console.log(`Downloading video: ${title}`);
        await downloadVideo(videoInfo, fileName);
        console.log(`Downloaded video: ${title}`);

        // Append downloaded video data to the file
        downloadedData.push({
            id: videoId,
            season: seasonCounter,
            episode: episodeCounter,
            url: videoUrl.trim(),
        });
        await writeFile(downloadedVideosFile,
JSON.stringify(downloadedData, null, 2));

        episodeCounter++;
    }
} catch (error) {
    console.error(`Error: ${error}`);
}
})();

```

Please write in English language.

I apologize for the confusion earlier. Based on your code, the poster should be generated in the same directory where you run the script, with the name `poster.jpg`. If the poster is not being generated, it could be due to the image generation process failing.

To debug the issue, I recommend adding some logging statements in the `generatePoster` function to see if the function is being called and if there are any errors during the image generation process:

```

javascript  Copy code

async function generatePoster(inputImage, outputImage) {
    console.log('Generating poster...'); // Add this line
    const image = sharp(inputImage);
    const metadata = await image.metadata();

    const fontSize = Math.round(metadata.width / 10);
    const text = `Season ${season}: ${title}`;

    await image
        .composite([
            {
                input: Buffer.from(`<svg xmlns="http://www.w3
                    metadata.width
                ">
                <text x="20" y="${
                    fontSize + 20
                }" font-size="${fontSize}" font-family="Arial"
                    ${text}
                </text>
                </svg>`),
            },
        ])
        .toFile(outputImage)

```

```
.catch((error) => { // Add this block for error h
  console.error('Error generating poster:', error
});

console.log('Poster generated:', outputImage); // A
}
```

These logging statements should help you identify if the `generatePoster` function is being called and if the poster is generated successfully. If there are any errors during the image generation process, the added `catch` block will log them.

If the issue persists, please let me know, and I'll help you investigate further.



cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker\$ node playlist\_download.mjs "https://www.youtube.com/playlist?list=PLTXrymoBNK\_YXyswBCIEOGM5YqBRRzOEC" negotiating comparing with previous downloads in: downloads/downloaded\_videos\_PLTXrymoBNK\_YXyswBCIEOGM5YqBRRzOEC.json Getting info for videoid: WChxbBSIWnQ WChxbBSIWnQ, 4VBjjkbEthY, TQHW7gGjrCQ Generating poster... Fontconfig error: Cannot load default config file: No such file: (null) Poster generated: Interviewing

Please write in English language.

It seems like the poster is generated, but there is a Fontconfig error during the process. This error is related to the underlying library used by `sharp` for image manipulation. The error message indicates that the default font configuration file cannot be loaded.



To fix this issue, you can try installing the `fontconfig` package if you are using a Linux-based operating system.

For Ubuntu/Debian:

```
bash
sudo apt-get update
sudo apt-get install fontconfig
```



For CentOS/RHEL/Fedora:

```
bash
sudo yum install fontconfig
```



After installing the fontconfig package, try running your script again. If the issue persists, you may need to configure the fontconfig library. For more information on configuring fontconfig, you can refer to the documentation [here](#).

Please let me know if this solves the issue or if you need further assistance.



I'm on a synology nas server

Please write in English language.

My apologies for not addressing that earlier. On a Synology NAS, you can use the `ipkg` package manager (also known as Entware) to install additional packages. First, you need to install the `ipkg` package manager if you haven't already.

1. Install Entware-ng package from SynoCommunity repository.  
To add the SynoCommunity repository to your Synology NAS, follow these steps:
  - a. Open the Package Center in your DSM (DiskStation Manager).
  - b. Click on Settings in the top right corner.
  - c. Go to the Package Sources tab.
  - d. Click Add and fill in the Name (e.g., SynoCommunity) and Location (<http://packages.synocommunity.com/>) fields.
  - e. Click OK to save the new package source.
2. Update the list of available packages:

bash

 Copy code

```
/opt/bin/ipkg update
```

3. Install the fontconfig package:

bash

 Copy code

```
/opt/bin/ipkg install fontconfig
```

4. After installing the fontconfig package, try running your script again.

Please let me know if this solves the issue or if you need further assistance.



Entware-ng seems to have been abandoned 5 years ago

Please write in English language.

I apologize for the outdated information. You're correct that Entware-ng is no longer maintained. Instead, you can use Entware, which is the active and maintained version. You can follow these steps to install Entware on your Synology NAS:

1. Install the "Task Scheduler" package from the Package Center if you haven't already.
2. Download the Entware installation script for your Synology NAS model from the following link:  
<https://github.com/Entware/Entware/wiki/Install-on-Synology-NAS>
3. Upload the downloaded script to your NAS using File Station or another preferred method.
4. Open Task Scheduler in DSM.
5. Create a new scheduled task:
  - a. Click "Create" > "Scheduled Task" > "User-defined script."
  - b. Give the task a name, such as "Entware Install."
  - c. In the "Task Settings" tab, set the "User" to "root."
  - d. In the "Run command" field, enter the following, replacing "/path/to/script.sh" with the actual path to the downloaded script on your NAS:

bash

 Copy code

```
/bin/bash /path/to/script.sh
```

6. Click "OK" to save the task.
7. Select the newly created task and click the "Run" button at the top of the window to run the script.

Once Entware is installed, you can use it to install the fontconfig package:

1. Update the list of available packages:

bash

 Copy code

```
/opt/bin/opkg update
```

## 2. Install the fontconfig package:

```
bash
```

[Copy code](#)

```
/opt/bin/opkg install fontconfig
```

After installing the fontconfig package, try running your script again.

Please let me know if this solves the issue or if you need further assistance.



Is there some other way besides entware?

Please write in English language.

If you prefer not to use Entware, another option is to use Docker to run your script in a container that has all the required dependencies, including fontconfig.



1. Install Docker from the Package Center on your Synology NAS if you haven't already.
2. Open Docker and go to the "Registry" tab. Search for a suitable Docker image, such as "node:14" or "node:its" (these are official Node.js images). Click "Download" to download the image to your NAS.
3. After the download is complete, switch to the "Image" tab, select the downloaded image, and click "Launch" to create a new container.
4. In the container creation wizard, configure the container to your needs. Be sure to set up volume mounts to map the required files and directories from your NAS to the container. For example, you can map the "/volume1/McCullohShare/Projects/video-maker" directory on your NAS to the "/app" directory in the container.
5. In the "Environment" tab, add an environment variable called "FONTCONFIG\_PATH" with the value "/etc/fonts".
6. Click "Apply" to create and start the container.
7. Connect to the container using a terminal:
  - a. Install the "Terminal & SNMP" package from the Package Center if you haven't already.
  - b. Open a terminal window and log in to your NAS using SSH.
  - c. Run the following command, replacing "container\_name" with the actual name of your Docker container:

```
bash
```

[Copy code](#)

```
sudo docker exec -it container_name /bin/bash
```

8. Once you are inside the container, navigate to the "/app" directory (or the directory you mapped to) and run your script:

```
bash
```

[Copy code](#)

```
cd /app
node playlist_download.mjs "https://www.youtube.com/p
```

Running your script in a Docker container should resolve the fontconfig issue. If you encounter any other issues or need further assistance, please let me know.



Is there some other way to change the generatePoster to generate the image that wouldn't require docker or entware or fontconfig?



Please write in English language.

Certainly! You can modify the `generatePoster` function to use a different library that doesn't require fontconfig. One such library is `sharp`, a high-performance image processing library for Node.js.

To use `sharp`, you'll need to install it first:

bash

 Copy code

```
npm install sharp
```

Next, replace the existing `generatePoster` function in your script with the following code:

javascript

 Copy code

```
import sharp from 'sharp';

async function generatePoster(videoInfo) {
  const width = 1920;
  const height = 1080;
  const backgroundColor = '#000000';
  const textColor = 'FFFFFF';
  const text = videoInfo.title.toUpperCase();

  const buffer = await sharp({
    create: {
      width,
      height,
      channels: 4,
      background: { r: 0, g: 0, b: 0, alpha: 1 },
    },
  })
  .composite([
    {
      input: Buffer.from(
        `<svg width="${width}" height="${height}">
          <rect x="0" y="0" width="${width}" height
            <text x="50%" y="50%" font-family="Arial,
              ${text}
            </text>
          </svg>`
      ),
      top: 0,
      left: 0,
    },
  ])
  .toBuffer();

  await fs.promises.writeFile(`posters/${videoInfo.vi
}
```

This implementation of `generatePoster` uses the `sharp` library to create a new image with the specified width, height, background color, and text color. It then overlays the video title on the image using an embedded SVG. Finally, it saves the resulting image as a PNG file.

Please note that this implementation uses the Arial font, which should be available on most systems. You can change the font family in the `font-family` attribute of the SVG text element if needed.

With this change, you should be able to generate posters without needing to install fontconfig, Docker, or Entware.



Why wouldn't that require fontconfig but the previous version would?

Please write in English language.

The original `generatePoster` function likely used a different library for image manipulation that relied on fontconfig for handling fonts. Fontconfig is a library designed to provide a centralized way to configure and customize font access for applications using scalable font technologies like FreeType, which is often used in image manipulation libraries.



In contrast, the `sharp` library is built on top of the libvips library, which has its own font handling mechanism and doesn't depend on fontconfig. By using `sharp`, you avoid the need for installing and configuring fontconfig on your system. This makes it a more portable solution, especially on systems like a Synology NAS where installing additional libraries can be more challenging.

The `sharp`-based implementation provided in the previous response uses an embedded SVG to create the text overlay. SVG is a vector graphics format that supports text rendering using system fonts, and in this case, the implementation specifies the Arial font, which is commonly available on most systems. Since `sharp` with embedded SVG handles fonts without relying on fontconfig, it's a more suitable solution for your particular use case.



So, would the following code require fontconfig? Right now the text is just displaying in the final poster as black rectangles...




```
async function generatePoster(inputImage, outputImage, title,
season) {
  console.log("Generating poster..."); // Add this line for
debugging
  const image = sharp(inputImage);
  const metadata = await image.metadata();

  const fontSize = Math.round(metadata.width / 10);
  const text = `Season ${season}: ${title}`;

  await image
    .composite([
      {
        input: Buffer.from(`<svg
xmlns="http://www.w3.org/2000/svg" width="${
          metadata.width
        }" height="${metadata.height}">
<text x="20" y="${
          fontSize + 20
        }" font-size="${fontSize}" font-family="Arial, sans-serif"
fill="white" stroke="black" stroke-width="2">
  ${text}
</text>
</svg>`),
      },
    ])
    .toFile(outputImage)
    .catch((error) => {
      // Add this block for error handling
      console.error("Error generating poster:", error);
    });

  console.log("Poster generated:", outputImage); // Add this line
for debugging
}
```

Please write in English language.

The code provided in your message does not require fontconfig.   

The text is generated using an embedded SVG, and the specified font family is "Arial, sans-serif". This way, the system should default to the Arial font or any available sans-serif font if Arial is not found.

However, you mentioned that the text is displayed as black rectangles in the final poster. This might be due to an issue with the font rendering on your Synology NAS system.

To address this issue, you can try the following steps:

1. Check if the Arial font is available on your Synology NAS system. If it's not available, you can try using a different font that is available on your system, or you can install the Arial font.
2. If the issue persists, it could be a problem with the SVG rendering. You can try creating a simple SVG file with some text using the same code structure and render it on your system to see if the issue is specific to the `sharp` library or the SVG itself.

Here's an example of a simple SVG file with text:

```
xml  Copy code

<svg xmlns="http://www.w3.org/2000/svg" width="800" height="100">
  <text x="20" y="100" font-size="40" font-family="Arial">
    Test Text
  </text>
</svg>
```

Save this as a `.svg` file and open it with an image viewer or web browser on your Synology NAS system. If the text renders correctly, then the issue might be with the `sharp` library or its interaction with the Synology NAS system. If the issue persists, it might be an issue with the SVG rendering on your system, and you may need to investigate further.

Please let me know if you need more assistance or if you encounter any other issues.



How can I check what fonts are available on my Synology NAS system? Or how can I install a font?

Please write in English language.

To check available fonts on your Synology NAS system or install a new font, follow these steps:

1. Access the command line interface (CLI) of your Synology NAS system using SSH. You can use an SSH client like PuTTY on Windows or the Terminal on macOS or Linux. To connect to your Synology NAS, use the following command:

```
css  Copy code

ssh username@your-nas-ip-address
```

Replace `username` with your Synology NAS username and `your-nas-ip-address` with the IP address of your Synology NAS.

2. Enter your password when prompted.
3. To list the available fonts on your system, use the following command:

 Copy code

```
fc-list
```

This command will show you a list of installed fonts on your Synology NAS system.

4. If you don't see the desired font (e.g., Arial) in the list, you'll need to install it. To do this, first download the font file (usually with the `.ttf` or `.otf` extension) from a trusted source. You can find Arial and other fonts on websites like <https://www.fontsquirrel.com> or <https://www.1001fonts.com>.
5. Transfer the downloaded font file to your Synology NAS system. You can use an FTP client like FileZilla or SCP (secure copy protocol) to transfer the file.
6. Once the font file is on your Synology NAS system, you'll need to create a font directory if it doesn't exist:

```
bash
```

[Copy code](#)

```
mkdir -p ~/.fonts
```

7. Move the font file to the `~/.fonts` directory:

```
bash
```

[Copy code](#)

```
mv /path/to/font-file.ttf ~/.fonts/
```

Replace `/path/to/font-file.ttf` with the path of the font file on your Synology NAS system.

8. Update the font cache:

[Copy code](#)

```
fc-cache -fv
```

Now, the font should be available on your Synology NAS system. You can confirm this by running the `fc-list` command again and looking for the font in the list.

After installing the font, you can try running your code again to see if the text renders correctly in the poster.



```
cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker$ fc-list
-sh: fc-list: command not found
```

Please write in English language.

My apologies for the confusion earlier. It appears that `fc-list` is not available on your Synology NAS system by default.

You can try using the `find` command to list available fonts on your Synology NAS system. Run the following command:

```
bash
```

[Copy code](#)

```
find / -type f \( -iname \*.ttf -o -iname \*.otf \) 2
```

This command will search for font files (with `.ttf` and `.otf` extensions) on your Synology NAS system and display the results.

If you cannot find the desired font, you can still install it by following steps 4-8 in my previous response. Once you have transferred the font file to the `~/.fonts` directory and updated

the font cache, the font should be available for use in your project.



Is there a way I can provide meta-data for a tv show to plex? Like, if I have a folder for a tv show called "negotiating", how do I provide the meta-data (show name, etc) as a file (instead of having to use the interface)?

Please write in English language.

Yes, you can provide custom metadata for your TV shows in Plex by using a local media assets agent. Here's a step-by-step guide on how to do it:

1. Create a metadata folder structure: In the same directory as your TV show, create a folder named "metadata." Inside that folder, create another folder with the same name as the TV show, in this case, "negotiating."
2. Add metadata files: Inside the "negotiating" folder, you can create files with metadata. For example, you can create a "poster.jpg" file for the show's poster image and a "background.jpg" file for the show's background image.
3. Create an NFO file: To provide more details about the show, create an XML file with the ".nfo" extension, such as "tvshow.nfo." In this file, you can add information like the show title, plot, actors, and more. Here's a basic example:

xml

Copy code

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"
<tvshow>
  <title>Negotiating</title>
  <plot>A TV show about the art of negotiation.</pl
  <genre>Drama</genre>
  <rating>8.5</rating>
</tvshow>
```

4. Enable the Local Media Assets agent in Plex: In the Plex Web App, go to Settings > Agents > Shows > Plex TV Series. Ensure that "Local Media Assets (TV)" is checked and move it to the top of the list using the drag handles.
5. Refresh your TV show library: Go to your TV show library, click the three-dot menu, and select "Scan Library Files." Plex should now use your custom metadata for the "negotiating" show.

Remember to follow the proper naming conventions for your TV show files and folders. For more information on this, check the Plex documentation:

[TV Show Naming](#)

[Local Media Assets - TV Shows](#)



For the XML you just gave me, how can I provide a content rating?

Please write in English language.

To provide a content rating for your TV show in the NFO file, you can use the `**<mpaa>**` tag. Here's an example of how to include it in your NFO file:

xml

Copy code

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"
<tvshow>
  <title>Negotiating</title>
  <plot>A TV show about the art of negotiation.</pl
```

```
<genre>Drama</genre>
<rating>8.5</rating>
<mpaa>TV-14</mpaa>
</tvshow>
```

In this example, I added the `**<mpaa>**` tag with a content rating of "TV-14." You can replace "TV-14" with the appropriate content rating for your show. Common content ratings include "TV-Y," "TV-Y7," "TV-G," "TV-PG," "TV-14," and "TV-MA." You can also use movie content ratings like "G," "PG," "PG-13," "R," or "NC-17" if it's more appropriate for your content.



Write a method that uses the `playlistTitle` we already made for the `generatePoster` method and writes that NFO for the show in the proper place.

Please write in English language.

Here's a method that creates the NFO file for a TV show using the `playlistTitle`, content rating, and other metadata. This method assumes that the NFO file should be placed in the same folder as the TV show episodes:

```
javascript Copy code

const fs = require('fs');
const path = require('path');

async function createShowNFO(playlistTitle, contentRa
const nfoContent = `<?xml version="1.0" encoding="U
<tvshow>
  <title>${playlistTitle}</title>
  <plot>A description of the show's plot.</plot>
  <genre>Genre of the show</genre>
  <rating>8.5</rating>
  <mpaa>${contentRating}</mpaa>
</tvshow>
`;

const nfoFilePath = path.join(showFolderPath, 'tvsh
try {
  await fs.promises.writeFile(nfoFilePath, nfoConte
  console.log(`NFO file created at ${nfoFilePath}`)
} catch (error) {
  console.error('Error creating NFO file:', error);
}

// Usage example:
createShowNFO('Negotiating', 'TV-14', '/path/to/your/
```

Replace the placeholder strings in the `**<plot>**` and `**<genre>**` tags with the appropriate plot description and genre for your show. You can also adjust the `**<rating>**` tag value if needed. To use this method, call `createShowNFO` with the `playlistTitle`, content rating, and the path to the TV show folder where the NFO file should be placed.



```
import { exec, execFile } from "child_process";
import util from "util";
import fs from "fs";
import path from "path";
import axios from "axios";
const execPromisified = util.promisify(exec);
const writeFile = util.promisify(fs.writeFile);
const readFile = util.promisify(fs.readFile);
import { pipeline } from "stream/promises";
import fetch from "node-fetch";

const playlistUrl = process.argv[2];
const folderName = process.argv[3];
```

```

const contentRating = process.argv[4] || "TV-14";
const cookiesFilePath = "cookies.txt";

if (fs.existsSync("downloads")) {
  fs.mkdirSync("downloads");
}

const execFilePromisified = (command, args, options = {}) => {
  return new Promise((resolve, reject) => {
    execFile(command, args, options, (error, stdout, stderr) => {
      if (error) {
        reject(error);
      } else {
        resolve({ stdout, stderr });
      }
    });
  });
};

const isValidJSON = (jsonString) => {
  try {
    JSON.parse(jsonString);
    return true;
  } catch (e) {
    return false;
  }
};

let downloadedVideosFile;
let outputDir;

if (playlistUrl.includes("playlist")) {
  const playlistId = new URL(playlistUrl).searchParams.get("list");
  downloadedVideosFile =
    `downloads/downloaded_videos_${playlistId}.json`;
  outputDir = `/volume1/McCullohShare/Plex/TV/${folderName}
? folderName : playlistId`;
} else if (playlistUrl.includes("videos")) {
  let channelName;

  try {
    ({ stdout: channelName } = await execPromisified(
      `yt-dlp --cookies ${cookiesFilePath} --get-filename --no-
playlist --output "%(uploader)s" "${playlistUrl}"`
    ));
    channelName = channelName.trim().split("\n")[0];
    console.log("channelName", channelName);
  } catch (error) {
    console.error(`Error getting channel name: ${error}`);
    process.exit(1);
  }

  downloadedVideosFile =
    `downloads/downloaded_videos_${channelName}.json`;
  outputDir = `/volume1/McCullohShare/Plex/TV/${folderName}
? folderName : channelName`;
} else {
  console.error("Invalid URL format. Please provide a playlist or
videos URL.");
  process.exit(1);
}

console.log("comparing with previous downloads in:",
downloadedVideosFile);

if (fs.existsSync(outputDir)) {
  fs.mkdirSync(outputDir);
}

async function downloadThumbnail(thumbnailUrl, outputPath) {
  try {
    console.log();
    console.log("Downloading thumbnail...", thumbnailUrl,
outputPath);

    const response = await axios({
      method: "GET",
      url: thumbnailUrl,
      responseType: "stream",
    });
  }

```

```

const writer = fs.createWriteStream(outputPath);
response.data.pipe(writer);

await new Promise((resolve, reject) => {
  writer.on("finish", resolve);
  writer.on("error", reject);
});

console.log();
console.log("Thumbnail downloaded:", outputPath);
} catch (error) {
  console.error("Error downloading thumbnail:",
error.message);
}
}

async function writeNfo(outputPath, videoInfo) {
  const nfoContent = `<?xml version="1.0" encoding="UTF-8"
standalone="yes" ?>
<episodedetails>
<title>${videoInfo.title}</title>
<plot>${videoInfo.description}</plot>
</episodedetails>
`;

  await writeFile(outputPath, nfoContent);
}

async function downloadVideo(videoInfo, filename) {
  return new Promise(async (resolve, reject) => {
    try {
      const response = await fetch(videoInfo.url);
      const totalBytes = parseInt(response.headers.get("content-
length"), 10);
      let downloadedBytes = 0;

      const progressBarLength = 30;

      const updateProgressBar = () => {
        const progressPercentage = (downloadedBytes /
totalBytes) * 100;
        const progressBarFilled = Math.floor(
          progressPercentage / (100 / progressBarLength)
        );
        const progressBarEmpty = progressBarLength -
progressBarFilled;
        const progressBar =
`[${"=".repeat(progressBarFilled)}${" ".repeat(
          progressBarEmpty
        )}]`;

        process.stdout.clearLine();
        process.stdout.cursorTo(0);
        process.stdout.write(
          `Downloading: ${progressBar}
${Math.floor(progressPercentage)}% `
        );
      };

      const fileStream = fs.createWriteStream(filename);
      response.body.on("data", (chunk) => {
        downloadedBytes += chunk.length;
        updateProgressBar();
      });

      await pipeline(response.body, fileStream);

      const thumbnailFilename = `${path.basename(filename,
.mp4)}.jpg`;
      const thumbnailOutputPath = path.join(outputDir,
thumbnailFilename);
      await downloadThumbnail(videoInfo.thumbnail,
thumbnailOutputPath);

      const nfoFilename = `${path.basename(filename,
.mp4)}.nfo`;
      const nfoOutputPath = path.join(outputDir, nfoFilename);
      await writeNfo(nfoOutputPath, videoInfo);

      console.log(); // Print a newline after the progress bar
      resolve();
    } catch (error) {
      reject(error);
    }
  });
}

```



```

    } catch (error) {
      fs.unlink(filename, () => {});
      reject(error);
    }
  });
}

const getPlaylistInfo = async (url) => {
  try {
    const { stdout: jsonInfo } = await execFilePromisified(
      "yt-dlp",
      ["--cookies", cookiesFilePath, "--skip-download", "--print-
json", url],
      { maxBuffer: 10 * 1024 * 1024 }
    );

    const jsonObjects = jsonInfo.split("\n").filter((obj) =>
obj.trim() !== "");
    let playlistInfo = null;

    for (const jsonObject of jsonObjects) {
      try {
        const parsedInfo = JSON.parse(jsonObject);

        playlistInfo = {
          id: parsedInfo.playlist_id || parsedInfo.channel_id,
          title: parsedInfo.playlist_title || parsedInfo.channel,
        };

        if (playlistInfo.id && playlistInfo.title) {
          break;
        }
      } catch (error) {
        console.error(`Error parsing JSON object: ${error}`);
      }
    }

    if (!playlistInfo || !playlistInfo.id || !playlistInfo.title) {
      throw new Error("Unable to get playlist or channel
information");
    }

    return playlistInfo;
  } catch (error) {
    console.error(`Error getting playlist info: ${error}`);
    return null;
  }
};

async function getVideoInfo(videoId) {
  const { stdout: jsonOutput } = await execPromisified(
    `yt-dlp --cookies ${cookiesFilePath} -j
https://www.youtube.com/watch?v=${videoId}`,
    { maxBuffer: 10 * 1024 * 1024 }
  );

  return JSON.parse(jsonOutput);
}

(async () => {
  try {
    const { stdout: ids } = await execPromisified(
      `yt-dlp -i -f best --cookies ${cookiesFilePath} --get-id
${playlistUrl}`
    );

    let downloadedData = [];
    if (fs.existsSync(downloadedVideosFile)) {
      const content = await readFile(downloadedVideosFile, "utf-
8");
      downloadedData = JSON.parse(content);
    }

    const videoIds = ids
      .split("\n")
      .filter((id) => id && !downloadedData.some((entry) =>
entry.id === id));

    if (playlistUrl.includes("videos")) {
      videoIds.reverse();
    }
  }

```

```

let seasonCounter;
let episodeCounter;
let lastyear;

if (downloadedData.length > 0) {
  const lastEntry = downloadedData[downloadedData.length
- 1];
  const lastEntryInfo = await getVideoInfo(lastEntry.id);
  const lastEntryUploadDate = lastEntryInfo.upload_date;
  lastyear = parseInt(lastEntryUploadDate.slice(0, 4), 10);
  seasonCounter = lastEntry.season;
  episodeCounter = lastEntry.episode + 1;
} else {
  console.log(` Getting info for videoid: ${videoids[0]} `,
videoids.join(", "));
  const firstVideoInfo = await getVideoInfo(videoids[0]);
  const firstVideoUploadDate = firstVideoInfo.upload_date;
  lastyear = parseInt(firstVideoUploadDate.slice(0, 4), 10);
  seasonCounter = 1;
  episodeCounter = 1;
}

// Get the name of the playlist as it exists on YouTube
const playlistInfo = await getPlaylistInfo(playlistUrl);
const playlistTitle = playlistInfo.title;
// CHATGPT: call a createShowNFO method here that saves
an nfo file for the playlist in the correct place for plex to find for
the show

for (const videoid of videoids) {
  console.log(` Getting info for videoid: ${videoid} `);
  const { stdout: videoUrl } = await execPromisified(
`yt-dlp -i -f best --cookies ${cookiesFilePath} --get-url
https://www.youtube.com/watch?v=${videoid} `,
{ maxBuffer: 10 * 1024 * 1024 }
);

  const videoInfo = await getVideoInfo(videoid);
  videoInfo.url = videoUrl;
  const title = videoInfo.title;
  const videoUploadDate = videoInfo.upload_date;

  const currentYear = parseInt(videoUploadDate.slice(0, 4),
10);
  if (currentYear > lastyear) {
    seasonCounter = seasonCounter += 1;
    episodeCounter = 1;
    lastyear = currentYear;
  }

  const fileName = `${outputDir}/${S}${seasonCounter
.toString()
.padStart(2,
"0"))E${episodeCounter.toString().padStart(2,
"0"))}-${title.replace(
/[\\/:.*?*<>|]/g,
"_"
)}.mp4`;




  console.log(` Downloading video: ${title} `);
  await downloadVideo(videoInfo, fileName);
  console.log(` Downloaded video: ${title} `);

  // Append downloaded video data to the file
  downloadedData.push({
    id: videoid,
    season: seasonCounter,
    episode: episodeCounter,
    url: videoUrl.trim(),
  });
  await writeFile(downloadedVideosFile,
JSON.stringify(downloadedData, null, 2));

  episodeCounter++;
}
} catch (error) {
  console.error(` Error: ${error} `);
}
})();

```

Please write in English language.

It seems like you have pasted your entire script in the message.   


However, I understand that you want to add the `createShowNFO` function to your current script. Here's the modified version of the `createShowNFO` function and its usage within your script:

1. Add the `fs` and `path` imports if they're not already in your script:

javascript  Copy code

```
import fs from "fs";
import path from "path";
```


2. Include the modified `createShowNFO` function in your script:

javascript  Copy code

```
async function createShowNFO(playlistTitle, contentRa
const nfoContent = `<?xml version="1.0" encoding="U
<tvshow>
  <title>${playlistTitle}</title>
  <plot>A description of the show's plot.</plot>
  <genre>Genre of the show</genre>
  <rating>8.5</rating>
  <mpaa>${contentRating}</mpaa>
</tvshow>
`;

const nfoFilePath = path.join(showFolderPath, 'tvsh
try {
  await fs.promises.writeFile(nfoFilePath, nfoConte
  console.log('NFO file created at ${nfoFilePath}')
} catch (error) {
  console.error('Error creating NFO file:', error);
}
}
```

3. Call the `createShowNFO` function after getting the `playlistTitle`:

javascript  Copy code

```
const playlistTitle = playlistInfo.title;

// Call the createShowNFO method here to save an NFO
await createShowNFO(playlistTitle, contentRating, out
```

This updated script now includes the `createShowNFO` function and its usage within the main script to create the NFO file for the TV show.



Earlier you told me that a "metadata" folder needed to be created and have another folder inside it with the same name as the tv show I was trying to add metadata for. Is that not correct? Can I just put a "tvshow.nfo" file inside of the tv show folder next to the video files?

Please write in English language.

Apologies for the confusion. I misunderstood your initial question, and my earlier response was incorrect.

Yes, you can place the "tvshow.nfo" file inside the TV show folder next to the video files. Plex will read the metadata from the "tvshow.nfo" file when it is located in the same folder as the video

files. There is no need to create a separate "metadata" folder in this case.

In the provided script, the `createShowNFO` function writes the "tvshow.nfo" file directly in the TV show folder. This should work correctly with Plex.



In the following code, is there a way to get the playlist description or channel description if one exists?

```
const getPlaylistInfo = async (url) => {
  try {
    const { stdout: jsonInfo } = await execFilePromisified(
      "yt-dlp",
      ["--cookies", cookiesFilePath, "--skip-download", "--print-json", url],
      { maxBuffer: 10 * 1024 * 1024 }
    );

    const jsonObjects = jsonInfo.split("\n").filter((obj) =>
obj.trim() !== "");
    let playlistInfo = null;

    for (const jsonObject of jsonObjects) {
      try {
        const parsedInfo = JSON.parse(jsonObject);

        playlistInfo = {
          id: parsedInfo.playlist_id || parsedInfo.channel_id,
          title: parsedInfo.playlist_title || parsedInfo.channel,
        };

        if (playlistInfo.id && playlistInfo.title) {
          break;
        }
      } catch (error) {
        console.error(`Error parsing JSON object: ${error}`);
      }
    }

    if (!playlistInfo || !playlistInfo.id || !playlistInfo.title) {
      throw new Error("Unable to get playlist or channel
information");
    }

    return playlistInfo;
  } catch (error) {
    console.error(`Error getting playlist info: ${error}`);
    return null;
  }
};
```

Please write in English language.

Yes, you can modify the `getPlaylistInfo` function to also return the playlist or channel description if it exists. You can achieve this by adding the description field to the `playlistInfo` object in the loop:

javascript

 Copy code

```
const getPlaylistInfo = async (url) => {
  try {
    const { stdout: jsonInfo } = await execFilePr
      "yt-dlp",
      ["--cookies", cookiesFilePath, "--skip-do
      { maxBuffer: 10 * 1024 * 1024 }
    );

    const jsonObjects = jsonInfo.split("\n").filt
    let playlistInfo = null;

    for (const jsonObject of jsonObjects) {
      try {
        const parsedInfo = JSON.parse(jsonObj

        playlistInfo = {
```

```

        id: parsedInfo.playlist_id || par
        title: parsedInfo.playlist_title
        description: parsedInfo.playlist_
    };

    if (playlistInfo.id && playlistInfo.t
        break;
    }
} catch (error) {
    console.error('Error parsing JSON obj
}
}

if (!playlistInfo || !playlistInfo.id || !pla
    throw new Error("Unable to get playlist o
}

return playlistInfo;
} catch (error) {
    console.error('Error getting playlist info: $
    return null;
}
};

```

By adding the `description` field, you'll now be able to access the playlist or channel description from the returned `playlistInfo` object.



It isn't picking up the data in my tvshow.nfo file. Is it supposed to be "tvshow.nfo" or should it be named after the show itself?

Please write in English language.

I apologize for the confusion. For Plex, the "tvshow.nfo" file should be named after the show itself. To do this, you should replace "tvshow.nfo" with the show name in the `createShowNFO` function you have created earlier. Make sure the NFO file has the same name as the TV show folder.

Here's the modified `createShowNFO` function:

```

javascript Copy code

async function createShowNFO(showFolderPath, showTitl
const showNFOContent = `<?xml version="1.0" encodin
<tvshow>
  <title>${showTitle}</title>
  <plot>${showDescription}</plot>
  <mpaa>${contentRating}</mpaa>
</tvshow>
`;

const nfoFileName = `${showTitle}.nfo`;
const nfoFilePath = path.join(showFolderPath, nfoFi
await writeFile(nfoFilePath, showNFOContent);
}

```

Now the NFO file will be named after the show itself, and Plex should be able to pick up the metadata correctly.



That isn't working. Plex isn't recognizing the nfo file for the show

Please write in English language.

I apologize for the confusion. It appears that Plex does not natively support NFO files for TV shows. Plex uses its own metadata agents to fetch metadata, and by default, it does not read NFO files for TV shows.

However, you can use a third-party metadata agent like "XBMCnfoTVImporter" to make Plex read NFO files for TV shows.

Here's how to install and configure the XBMCnfoTVImporter agent:

1. Download the latest release of XBMCnfoTVImporter from the GitHub repository:  
<https://github.com/gboudreau/XBMCnfoTVImporter.bundle/releases>
2. Extract the downloaded ZIP file.
3. Rename the extracted folder to "XBMCnfoTVImporter.bundle".
4. Move the "XBMCnfoTVImporter.bundle" folder to the Plex Media Server's plugin directory. The location of the plugin directory varies depending on your operating system:
  - Windows: `%LOCALAPPDATA%\Plex Media Server\Plugins`
  - macOS: `~/Library/Application Support/Plex Media Server/Plug-ins`
  - Linux: `\$PLEX\_HOME/Library/Application Support/Plex Media Server/Plug-ins`
5. Restart the Plex Media Server.
6. In the Plex Web App, go to Settings > Server > Agents > Shows.
7. Select "TheTVDB" or "The Movie Database" (whichever you prefer), and move "XBMCnfoTVImporter" to the top of the list of metadata agents.
8. Click the gear icon next to "XBMCnfoTVImporter" to configure its settings. Make sure the "Enable for All Libraries" option is enabled.
9. Save the settings and update your TV show library in Plex.

Now, Plex should read the NFO files for your TV shows using the XBMCnfoTVImporter metadata agent.

Please note that this is a third-party plugin, and its functionality is not guaranteed by Plex.



Plex plugins have been discontinued and don't work anymore. Is there a way to embed metadata into the .mp4 file itself?

Please write in English language.

---

Yes, you can embed metadata directly into the MP4 files using a tool called "AtomicParsley." AtomicParsley is a command-line utility that allows you to read and write metadata in MPEG-4 files.

Here's how to use AtomicParsley to embed metadata into an MP4 file:

1. Download and install AtomicParsley from the official website:  
<http://atomicparsley.sourceforge.net/>
2. To embed metadata into an MP4 file, use the following command:

css

 Copy code

```
AtomicParsley input.mp4 --artist "Artist Name" --titl
```

Replace `input.mp4` with the path to your MP4 file, and set the appropriate metadata values.

3. AtomicParsley will create a new file with the metadata embedded, and the original file will remain unchanged. If you want to overwrite the original file, include the `--overWrite` option at the end of the command.

Here's an example of how to use AtomicParsley in the context of your current script:

javascript

 Copy code

```
async function embedMetadata(videoPath, metadata) {
  const { title, artist, album, year, tracknum, genre

  const command = `AtomicParsley "${videoPath}" --art

  try {
    await execPromisified(command);
  } catch (error) {
    console.error(`Error embedding metadata: ${error}`
  }
}
```

You can call this function after downloading the video, providing the path to the video and an object containing the metadata you want to embed.

Please note that Plex may still prioritize its own metadata agents over the embedded metadata in the MP4 file. To ensure that Plex uses the embedded metadata, you might need to adjust the settings for metadata agents in Plex, or modify the filenames to include relevant information (e.g., title, season, and episode number).



How do I use AtomicParsley on plex?

Please write in English language.

AtomicParsley is not a Plex plugin or tool, but it can be used to add metadata to your MP4 files before importing them into Plex. Plex should then recognize and display the embedded metadata when you add the video files to your library.



To use AtomicParsley for your MP4 files before adding them to Plex, follow these steps:

1. Download and install AtomicParsley from the official website:  
<http://atomicparsley.sourceforge.net/>
2. Embed metadata into your MP4 files using AtomicParsley



Sorry, how do I use it on a synology server?

Please write in English language.

To use AtomicParsley on a Synology NAS, you'll need to access the command line using SSH and install the software. Here's how to do it:

1. Enable SSH on your Synology NAS:
  - a. Open the Synology DiskStation Manager (DSM) web interface.
  - b. Go to Control Panel > Terminal & SNMP.
  - c. Check the box next to "Enable SSH service" and set the port (default is 22).
  - d. Click "Apply" to save the settings.
2. Connect to your Synology NAS using an SSH client like PuTTY (Windows) or Terminal (Mac):
  - a. Open the SSH client.
  - b. Enter the IP address of your Synology NAS, the port number you set earlier, and your login credentials (same as your DSM login).
3. Install `ipkg` package manager (if not already installed):
  - a. Follow the instructions in this link to install `ipkg` on your Synology NAS:  
<https://github.com/OneCDOnly/shellinabox/blob/master/synology/Bootstrap%20Installation.md>

4. Update ``ipkg`` and install AtomicParsley:
  - a. Run the following command to update the package list:  
``ipkg update``
  - b. Install AtomicParsley using the following command: ``ipkg install atomicparsley``
5. After the installation is complete, you can use AtomicParsley to embed metadata into your MP4 files.

Example usage of AtomicParsley:

css

 Copy code

```
AtomicParsley /path/to/video.mp4 --title "Example Tit
```

Replace the file paths and metadata with the appropriate values for your video files. Once you've embedded the metadata, you can add the video files to your Plex library. Plex should recognize the embedded metadata.

Remember to replace the example metadata with the metadata relevant to your video files.



Is there an alternative to AtomicParsley that would run more easily on a synology server?

Please write in English language.

FFmpeg is a versatile tool that can be used to manipulate video and audio files, including embedding metadata. It is widely used and should run more easily on a Synology server. Here's how to install and use FFmpeg on your Synology NAS:



1. Install FFmpeg using the Synology Package Center:
  - a. Open the DiskStation Manager (DSM) web interface.
  - b. Go to the Package Center and search for "FFmpeg."
  - c. Click "Install" to install the FFmpeg package.
2. Connect to your Synology NAS using an SSH client like PuTTY (Windows) or Terminal (Mac), as described in the previous answer.
3. Use FFmpeg to embed metadata into your MP4 files. Here's an example command:

lua

 Copy code

```
ffmpeg -i /path/to/input.mp4 -c copy -metadata title=
```

Replace the file paths and metadata with the appropriate values for your video files. Once you've embedded the metadata, you can add the video files to your Plex library. Plex should recognize the embedded metadata.

Remember to replace the example metadata with the metadata relevant to your video files.

Note that this command will not embed artwork. If you need to embed artwork as well, you can use the following command:

lua

 Copy code

```
ffmpeg -i /path/to/input.mp4 -i /path/to/cover.jpg -c
```

Replace the file paths and metadata with the appropriate values for your video files, including the path to the cover image.



how can I modify that ffmpeg script to provide episode description as well as title?



Please write in English language.

Unfortunately, FFmpeg does not have built-in support for embedding episode descriptions directly into MP4 files. However, you can include the description in the "comment" metadata field as a workaround. Keep in mind that Plex may not recognize this field as the episode description.



To add a description using the "comment" metadata field, modify the FFmpeg command as follows:

bash

Copy code

```
ffmpeg -i /path/to/input.mp4 -c copy -metadata title=
```

Replace "This is an example episode description." with the actual description of the episode.

Please note that this is not an ideal solution, and there's no guarantee that Plex will recognize the "comment" metadata field as the episode description.



OK, let's remove the nfo stuff from this script and use ffmpeg to embed the episode title and year:

```
import { exec, execFile } from "child_process";
import util from "util";
import fs from "fs";
import path from "path";
import axios from "axios";
const execPromisified = util.promisify(exec);
const writeFile = util.promisify(fs.writeFile);
const readFile = util.promisify(fs.readFile);
import { pipeline } from "stream/promises";
import fetch from "node-fetch";

const playlistUrl = process.argv[2];
const folderName = process.argv[3];
const cookiesFilePath = "cookies.txt";

if (!fs.existsSync("downloads")) {
  fs.mkdirSync("downloads");
}

const execFilePromisified = (command, args, options = {}) => {
  return new Promise((resolve, reject) => {
    execFile(command, args, options, (error, stdout, stderr) => {
      if (error) {
        reject(error);
      } else {
        resolve({ stdout, stderr });
      }
    });
  });
};

let downloadedVideosFile;
let outputDir;

if (playlistUrl.includes("playlist")) {
  const playlistId = new URL(playlistUrl).searchParams.get("list");
  downloadedVideosFile =
    `downloads/downloaded_videos_${playlistId}.json`;
  outputDir = ` /volume1/McCullohShare/Plex/TV/${folderName}
? folderName : playlistId`;
} else if (playlistUrl.includes("videos")) {
  let channelName;

  try {
    ({ stdout: channelName } = await execPromisified(
      `yt-dlp --cookies ${cookiesFilePath} --get-filename --no-
playlist --output "%(uploader)s" "${playlistUrl}"`
    ));
    channelName = channelName.trim().split("\n")[0];
  }
```

```

        console.log("channelName", channelName);
    } catch (error) {
        console.error(` Error getting channel name: ${error} `);
        process.exit(1);
    }

    downloadedVideosFile =
`downloads/downloaded_videos_${channelName}.json`;
    outputDir = `/volume1/McCullohShare/Plex/TV/${folderName}
? folderName : channelName`;
} else {
    console.error("Invalid URL format. Please provide a playlist or
videos URL.");
    process.exit(1);
}

console.log("comparing with previous downloads in: ",
downloadedVideosFile);

if (fs.existsSync(outputDir)) {
    fs.mkdirSync(outputDir);
}

async function downloadThumbnail(thumbnailUrl, outputPath) {
    try {
        console.log();
        console.log("Downloading thumbnail...", thumbnailUrl,
outputPath);

        const response = await axios({
            method: "GET",
            url: thumbnailUrl,
            responseType: "stream",
        });

        const writer = fs.createWriteStream(outputPath);
        response.data.pipe(writer);

        await new Promise((resolve, reject) => {
            writer.on("finish", resolve);
            writer.on("error", reject);
        });

        console.log();
        console.log("Thumbnail downloaded:", outputPath);
    } catch (error) {
        console.error("Error downloading thumbnail:",
error.message);
    }
}

async function writeNfo(outputPath, videoInfo) {
    const nfoContent = `<?xml version="1.0" encoding="UTF-8"
standalone="yes" ?>
<episodedetails>
<title>${videoInfo.title}</title>
<plot>${videoInfo.description}</plot>
</episodedetails>
`;

    await writeFile(outputPath, nfoContent);
}

async function downloadVideo(videoInfo, filename) {
    return new Promise(async (resolve, reject) => {
        try {
            const response = await fetch(videoInfo.url);
            const totalBytes = parseInt(response.headers.get("content-
length"), 10);
            let downloadedBytes = 0;

            const progressBarLength = 30;

            const updateProgressBar = () => {
                const progressPercentage = (downloadedBytes /
totalBytes) * 100;
                const progressBarFilled = Math.floor(
                    progressPercentage / (100 / progressBarLength)
                );
                const progressBarEmpty = progressBarLength -
progressBarFilled;

```

```

        const progressBar =
        `[${"=".repeat(progressBarFilled)}${" "}.repeat(
            progressBarEmpty
        )]`;

        process.stdout.clearLine();
        process.stdout.cursorTo(0);
        process.stdout.write(
            `Downloading: ${progressBar}
        ${Math.floor(progressPercentage)}% `
        );
    };

    const fileStream = fs.createWriteStream(filename);
    response.body.on("data", (chunk) => {
        downloadedBytes += chunk.length;
        updateProgressBar();
    });

    await pipeline(response.body, fileStream);

    const thumbnailFilename = `${path.basename(filename,
        ".mp4")}.jpg`;
    const thumbnailOutputPath = path.join(outputDir,
        thumbnailFilename);
    await downloadThumbnail(videoInfo.thumbnail,
        thumbnailOutputPath);

    const nfoFilename = `${path.basename(filename,
        ".mp4")}.nfo`;
    const nfoOutputPath = path.join(outputDir, nfoFilename);
    await writeNfo(nfoOutputPath, videoInfo);

    console.log(); // Print a newline after the progress bar
    resolve();
} catch (error) {
    fs.unlink(filename, () => {});
    reject(error);
}
});
}

const getPlaylistInfo = async (url) => {
    try {
        const { stdout: jsonInfo } = await execFilePromisified(
            "yt-dlp",
            ["--cookies", cookiesFilePath, "--skip-download", "--print-
            json", url],
            { maxBuffer: 10 * 1024 * 1024 }
        );

        const jsonObjects = jsonInfo.split("\n").filter((obj) =>
            obj.trim() !== "");
        let playlistInfo = null;

        for (const jsonObject of jsonObjects) {
            try {
                const parsedInfo = JSON.parse(jsonObject);

                playlistInfo = {
                    id: parsedInfo.playlist_id || parsedInfo.channel_id,
                    title: parsedInfo.playlist_title || parsedInfo.channel,
                    description: parsedInfo.playlist_description ||
                    parsedInfo.channel_description,
                };

                if (playlistInfo.id && playlistInfo.title) {
                    break;
                }
            } catch (error) {
                console.error(`Error parsing JSON object: ${error}`);
            }
        }

        if (!playlistInfo || !playlistInfo.id || !playlistInfo.title) {
            throw new Error("Unable to get playlist or channel
            information");
        }

        return playlistInfo;
    } catch (error) {

```

```

        console.error( ` Error getting playlist info: ${error} ` );
        return null;
    }
};

async function getVideoInfo(videoid) {
    const { stdout: jsonOutput } = await execPromisified(
        `yt-dlp --cookies ${cookiesFilePath} -j
https://www.youtube.com/watch?v=${videoid} `,
        { maxBuffer: 10 * 1024 * 1024 }
    );

    return JSON.parse(jsonOutput);
}

(async () => {
    try {
        const { stdout: ids } = await execPromisified(
            `yt-dlp -i -f best --cookies ${cookiesFilePath} --get-id
${playlistUrl} `
        );

        let downloadedData = [];
        if (fs.existsSync(downloadedVideosFile)) {
            const content = await readFile(downloadedVideosFile, "utf-
8");
            downloadedData = JSON.parse(content);
        }

        const videoids = ids
            .split("\n")
            .filter((id) => id && !downloadedData.some((entry) =>
entry.id === id));

        if (playlistUrl.includes("videos")) {
            videoids.reverse();
        }
        let seasonCounter;
        let episodeCounter;
        let lastyear;

        if (downloadedData.length > 0) {
            const lastEntry = downloadedData[downloadedData.length
- 1];
            const lastEntryInfo = await getVideoInfo(lastEntry.id);
            const lastEntryUploadDate = lastEntryInfo.upload_date;
            lastyear = parseInt(lastEntryUploadDate.slice(0, 4), 10);
            seasonCounter = lastEntry.season;
            episodeCounter = lastEntry.episode + 1;
        } else {
            console.log( ` Getting info for videoid: ${videoids[0]} ` ,
videoids.join(", ") );
            const firstVideoInfo = await getVideoInfo(videoids[0]);
            const firstVideoUploadDate = firstVideoInfo.upload_date;
            lastyear = parseInt(firstVideoUploadDate.slice(0, 4), 10);
            seasonCounter = 1;
            episodeCounter = 1;
        }

        for (const videoid of videoids) {
            console.log( ` Getting info for videoid: ${videoid} ` );
            const { stdout: videoUrl } = await execPromisified(
                `yt-dlp -i -f best --cookies ${cookiesFilePath} --get-url
https://www.youtube.com/watch?v=${videoid} `,
                { maxBuffer: 10 * 1024 * 1024 }
            );

            const videoInfo = await getVideoInfo(videoid);
            videoInfo.url = videoUrl;
            const title = videoInfo.title;
            const videoUploadDate = videoInfo.upload_date;

            const currentYear = parseInt(videoUploadDate.slice(0, 4),
10);
            if (currentYear > lastyear) {
                seasonCounter = seasonCounter += 1;
                episodeCounter = 1;
                lastyear = currentYear;
            }

            const fileName = ` ${outputDir}/${S}${seasonCounter

```

```

        .toString()
        .padStart(2,
"0"))E$(episodeCounter.toString().padStart(2,
"0"))-${title.replace(
    /\[.*?:.*?<>\/]/g,
    "-")
    }).mp4`;

    console.log(` Downloading video: ${title} `);
    await downloadVideo(videoInfo, fileName);
    console.log(` Downloaded video: ${title} `);

    // Append downloaded video data to the file
    downloadedData.push({
        id: videoId,
        season: seasonCounter,
        episode: episodeCounter,
        url: videoUrl.trim(),
    });
    await writeFile(downloadedVideosFile,
JSON.stringify(downloadedData, null, 2));

    episodeCounter++;
    }
} catch (error) {
    console.error(`Error: ${error}`);
}
})();

```

Please write in English language.

I have removed the NFO-related code and modified the script to use FFmpeg to embed the episode title and year into the .mp4 file. Here's the updated script:

```

javascript  Copy code

import { exec, execFile } from "child_process";
import util from "util";
import fs from "fs";
import path from "path";
import axios from "axios";
const execPromisified = util.promisify(exec);
const writeFile = util.promisify(fs.writeFile);
const readFile = util.promisify(fs.readFile);
import { pipeline } from "stream/promises";
import fetch from "node-fetch";

const playlistUrl = process.argv[2];
const folderName = process.argv[3];
const cookiesFilePath = "cookies.txt";

if (!fs.existsSync("downloads")) {
    fs.mkdirSync("downloads");
}

const execFilePromisified = (command, args, options =
return new Promise((resolve, reject) => {
    execFile(command, args, options, (error, stdo
        if (error) {
            reject(error);
        } else {
            resolve({ stdout, stderr });
        }
    });
});
});

let downloadedVideosFile;
let outputDir;

if (playlistUrl.includes("playlist")) {
    const playlistId = new URL(playlistUrl).searchPar
    downloadedVideosFile = `downloads/downloaded_vide
    outputDir = `volume1/McCullohShare/Plex/TV/${fol
} else if (playlistUrl.includes("videos")) {
    let channelName;

```

```

    try {
      ({ stdout: channelName } = await execPromisif
        `yt-dlp --cookies ${cookiesFilePath} --ge
      );
      channelName = channelName.trim().split("\n")[
        console.log("channelName", channelName);
      } catch (error) {
        console.error(`Error getting channel name: ${
          process.exit(1);
        }
      }

      downloadedVideosFile = `downloads/downloaded_vide
      outputDir = `/volume1/McCullohShare/Plex/TV/${fol
    } else {
      console.error("Invalid URL format. Please provide
      process.exit(1);
    }

    console.log("comparing with previous downloads in: ",

    if (!fs.existsSync(outputDir)) {
      fs.mkdirSync(outputDir);
    }

    async function downloadThumbnail(thumbnailUrl, output
    try {
      console.log();
      console.log("Downloading thumbnail...", thumb

      const response = await axios({
        method: "GET",
        url: thumbnailUrl,
        responseType: "stream",
      });

      const writer = fs.createWriteStream(outputPat
      response.data.pipe(writer);

      await new Promise((resolve, reject) => {
        writer.on("finish", resolve);
        writer.on("error", reject);
      });

      console.log();
      console.log("Thumbnail downloaded:", outputPa
    } catch (error) {
      console.error("Error downloading thumbnail:",
    }
  }

  async function downloadVideo(videoInfo, filename) {
    return new Promise(async (resolve, reject) => {
      try {
        const response = await fetch(videoInfo.ur
        const totalBytes = parseInt(response.head
        let downloadedBytes = 0;

        const progressBarLength = 30;

        const updateProgressBar = () => {
          const progressPercentage = (downloade
          const progressBarFilled = Math

```



continue

Please write in English language.

Sure, here's the continuation of the script:

```

javascript Copy code

    .floor(
      progressPercentage / (100 / progr
    );
    const progressBarEmpty = progressBarL
    const progressBar = `${"=".repeat(pr
    progressBarEmpty

```

```

    ]];

    process.stdout.clearLine();
    process.stdout.cursorTo(0);
    process.stdout.write(
        `Downloading: ${progressBar} ${Ma
    });
};

const tempFilename = `${filename}.temp`;
const fileStream = fs.createWriteStream(t
response.body.on("data", (chunk) => {
    downloadedBytes += chunk.length;
    updateProgressBar();
});

await pipeline(response.body, fileStream)

console.log(); // Print a newline after t

const metadataArgs = [
    "-i",
    tempFilename,
    "-c",
    "copy",

```

🔄 Regenerate response

Send a message.



ChatGPT may produce inaccurate information about people, places, or facts. [ChatGPT Mar 23 Version](#)

```

await execFilePromisified("ffmpeg", metad

fs.unlinkSync(tempFilename);

    resolve();
} catch (error) {
    fs.unlink(filename, () => {});
    reject(error);
}
});

// The rest of the script remains unchanged.

```

I have added a temporary filename and modified the `downloadVideo` function to first download the video to the temporary file. After that, the script uses FFmpeg to copy the video from the temporary file to the final output file while embedding the metadata (title and year) into the .mp4 file. Finally, the temporary file is deleted.

Please note that you need to have FFmpeg installed and available in the PATH on your Synology server for this script to work properly.



I'm getting the following error:

```

cmcculloh@McCullohNas:/volume1/McCullohShare/Projects/video-maker$
node playlist_download.mjs "https://www.youtube.com/playlist?
list=PLEzsuLYKNT8teSpGB_jd_1Rm1JseHAb1M" scott-bradlee
comparing with previous downloads in:
downloads/downloaded_videos_PLEzsuLYKNT8teSpGB_jd_1Rm1JseHAb1M.json
Error in main loop: Error: Command failed: yt-dlp -i --cookies cookies.txt --get-id
https://www.youtube.com/playlist?
list=PLEzsuLYKNT8teSpGB_jd_1Rm1JseHAb1M
ERROR: [youtube] jPPtXS4a9m4: This live stream recording is not available.

```

The error you're encountering is because one of the videos in the playlist is a live stream recording that is no longer available. You can add the `--ignore-errors` option to the `yt-dlp` command in the script to skip videos that are unavailable or have issues.

Here's the modified line that you can replace in your script:

 Copy code

This change should allow the script to continue running and process the rest of the videos in the playlist even if some of them are unavailable.



Getting info for videoid: Bp0F9SF3RmU  
 Downloading video: 'Twinkle, Twinkle' 24 Genre Piano Variations  
 by Scott Bradlee

Downloading thumbnail...  
[https://i.ytimg.com/vi\\_webp/Bp0F9SF3RmU/maxresdefault.webp/volume1/McCullohShare/Plex/TV/scott-bradlee/S01E88-'Twinkle, Twinkle' 24 Genre Piano Variations by Scott Bradlee.jpg](https://i.ytimg.com/vi_webp/Bp0F9SF3RmU/maxresdefault.webp/volume1/McCullohShare/Plex/TV/scott-bradlee/S01E88-'Twinkle, Twinkle' 24 Genre Piano Variations by Scott Bradlee.jpg)

Thumbnail downloaded: /volume1/McCullohShare/Plex/TV/scott-bradlee/S01E88-'Twinkle, Twinkle' 24 Genre Piano Variations by Scott Bradlee.jpg

```

adding metadata
error downloading 'Twinkle, Twinkle' 24 Genre Piano Variations by
Scott Bradlee: Error: Command failed: ffmpeg -i
/volume1/McCullohShare/Plex/TV/scott-bradlee/S01E88-
'Twinkle, Twinkle' 24 Genre Piano Variations by Scott
Bradlee.mp4.temp -c copy -metadata title='Twinkle, Twinkle' 24
Genre Piano Variations by Scott Bradlee -metadata
year=undefined /volume1/McCullohShare/Plex/TV/scott-
bradlee/S01E88-'Twinkle, Twinkle' 24 Genre Piano Variations by
Scott Bradlee.mp4
ffmpeg version 4.1.8 Copyright (c) 2000-2021 the FFmpeg
developers
built with gcc 8.5.0 (GCC)
configuration: --prefix=/usr --incdir=${prefix}/include/ffmpeg' -
-arch=i686 --target-os=linux --cross-prefix=/usr/local/x86_64-
pc-linux-gnu/bin/x86_64-pc-linux-gnu- --enable-cross-compile
--enable-optimizations --enable-pic --enable-gpl --enable-shared
--disable-static --disable-stripping --enable-version3 --enable-
encoders --enable-threads --disable-protocols --disable-
protocol=rtsp --enable-protocol=file --enable-protocol=pipes --
disable-muxer=image2 --disable-muxer=image2pipe --disable-
swscale-alpha --disable-fplay --disable-ffprobe --disable-doc --
disable-devices --disable-bzlib --disable-altivec --enable-
libopencore-amrnb --enable-libopencore-amrwb --enable-
libmp3lame --disable-vaapi --disable-cuvid --disable-nvenc --
disable-decoder=aac --disable-decoder=aac.fixed --disable-
encoder=aac --disable-decoder=amrnb --disable-decoder=ac3 --
disable-decoder=ac3_fixed --disable-encoder=zmbv --disable-
encoder=dca --disable-decoder=dca --disable-encoder=ac3 --
disable-encoder=ac3_fixed --disable-encoder=eac3 --disable-
decoder=eac3 --disable-encoder=truehd --disable-
decoder=truehd --disable-encoder=hevc_vaapi --disable-
decoder=hevc --disable-muxer=hevc --disable-demuxer=hevc --
disable-parser=hevc --disable-bsf=hevc_mp4toannexb --
x86asmext=yasm --cc=/usr/local/x86_64-pc-linux-
gnu/bin/x86_64-pc-linux-gnu-wrap-gcc --enable-yasm --enable-
libx264 --enable-encoder=libx264
libavutil 56. 22.100 / 56. 22.100
libavcodec 58. 35.100 / 58. 35.100
libavformat 58. 20.100 / 58. 20.100
libavdevice 58. 5.100 / 58. 5.100
libavfilter 7. 40.101 / 7. 40.101
libswscale 5. 3.100 / 5. 3.100
libswresample 3. 3.100 / 3. 3.100
libpostproc 55. 3.100 / 55. 3.100
/volume1/McCullohShare/Plex/TV/scott-bradlee/S01E88-
'Twinkle, Twinkle' 24 Genre Piano Variations by Scott
Bradlee.mp4.temp: Invalid data found when processing input

```

Downloaded video: 'Twinkle, Twinkle' 24 Genre Piano Variations  
by Scott Bradlee



It seems there is an issue with the downloaded video file, which is causing the error when running the `ffmpeg` command. The error message states: "Invalid data found when processing input." This indicates that the input video file might be corrupt or incomplete.

To address this issue, you can try the following steps:

1. Delete the temporary video file  
``/volume1/McCullohShare/Plex/TV/scott-bradlee/S01E88-'Twinkle, Twinkle' 24 Genre Piano Variations by Scott Bradlee.mp4.temp``.
2. Re-run the script to attempt downloading the video again. It's possible that the download was interrupted or failed previously, which led to a corrupt video file.

If the issue persists, you may need to investigate further to determine whether the video file is indeed corrupted, or if there are other issues with the download process.

---