Walkthrough

Why this code was created?

To download videos from Mediathek that can’t be accessed normally without some vpn extension helper. In this case I used this series as guinea pig [Falk] (https://www.fernsehserien.de/falk/episodenguide).

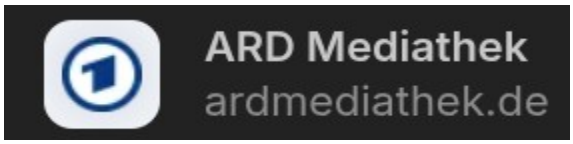
How this code works?

First install a browser, in my case Chromium (it’s an open source browser based Chrome but without the bad parts of it). You can change and use what browser you like.

After installer you need to download some free VPN extension for you browser (setupVPN in my case).

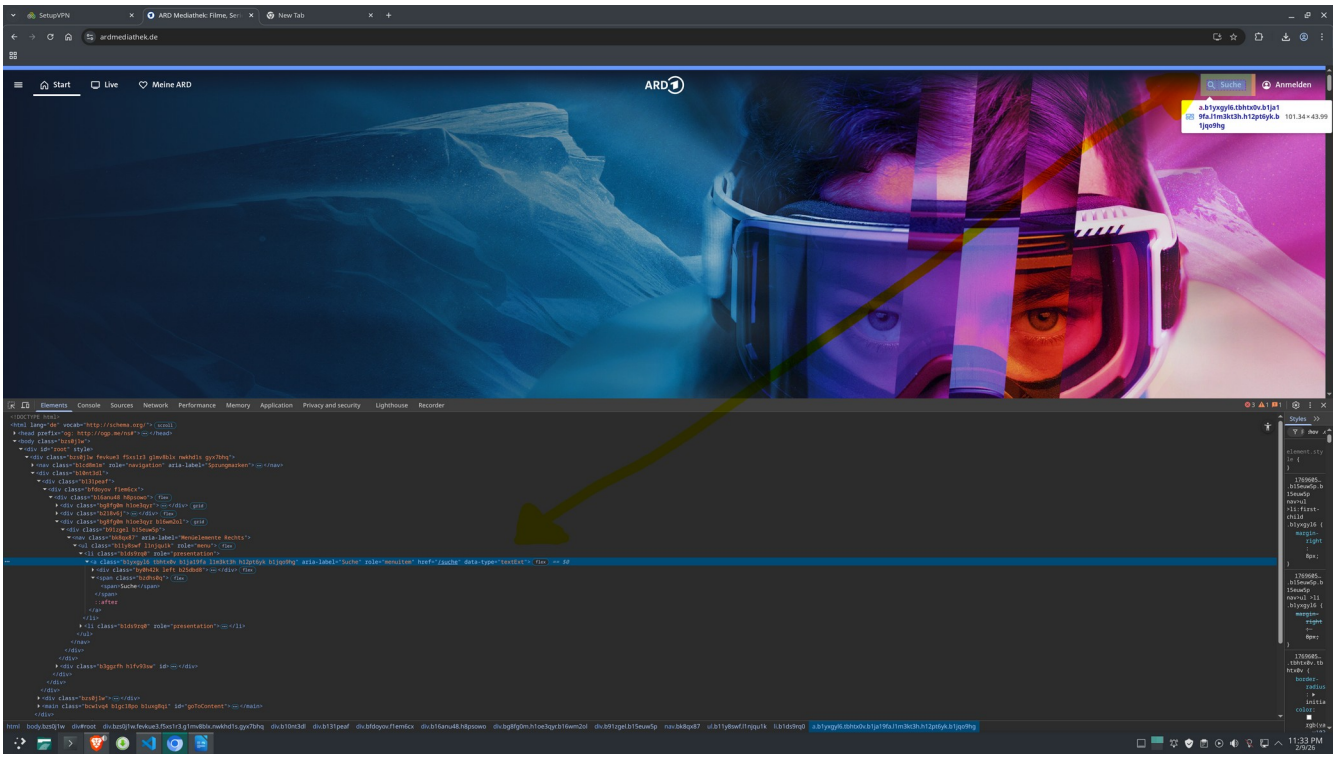
MAKE SURE THAT THE VPN EXTENSION IS OPENED AND WORKS BEFORE YOU OPEN MEDIATHEK OTHERWISE THE SIDE WILL SAY “sORrY tHe ViDeO iS nOt AvAiLaBLe iN YoUr cOuNtRy”.

When you open the Mediathek, in the search button you type the series that is available to watch (as sometimes they might delete and put the series back after a while).



Before seeing the code let’s see how the site looks like and see how to save a video.

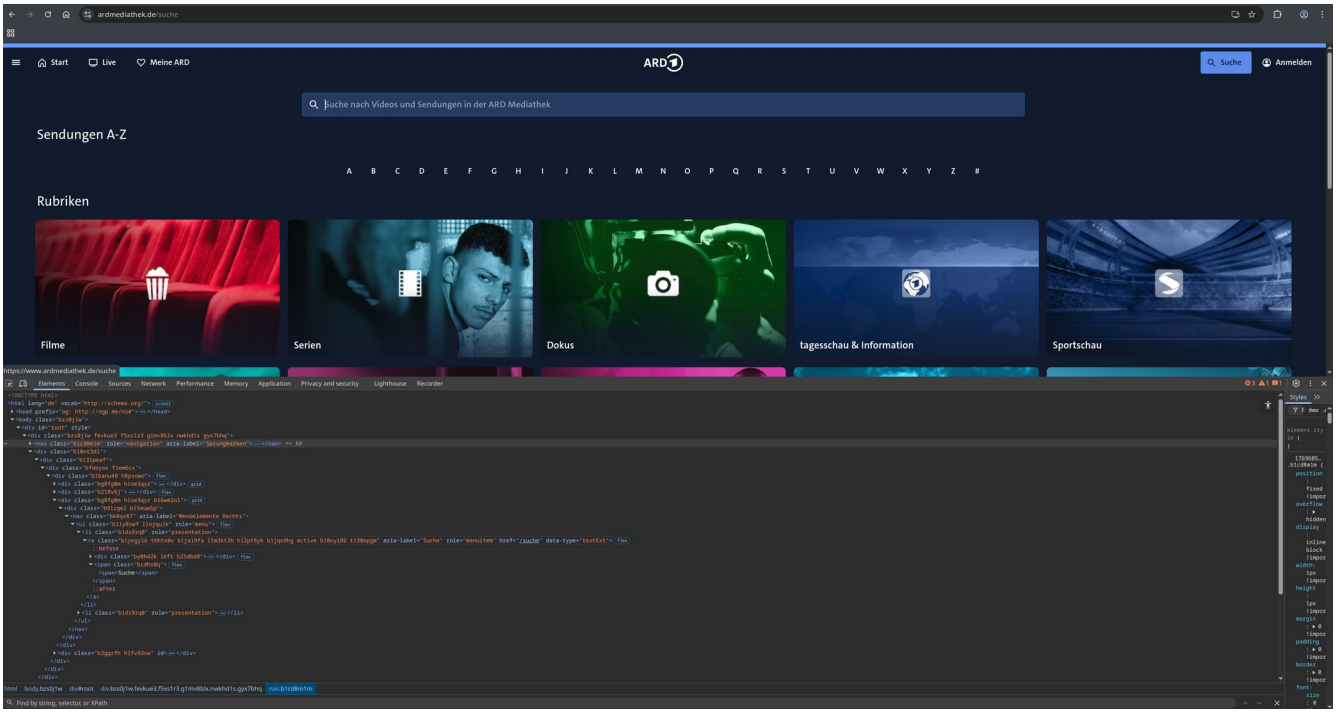
Manual walktrough



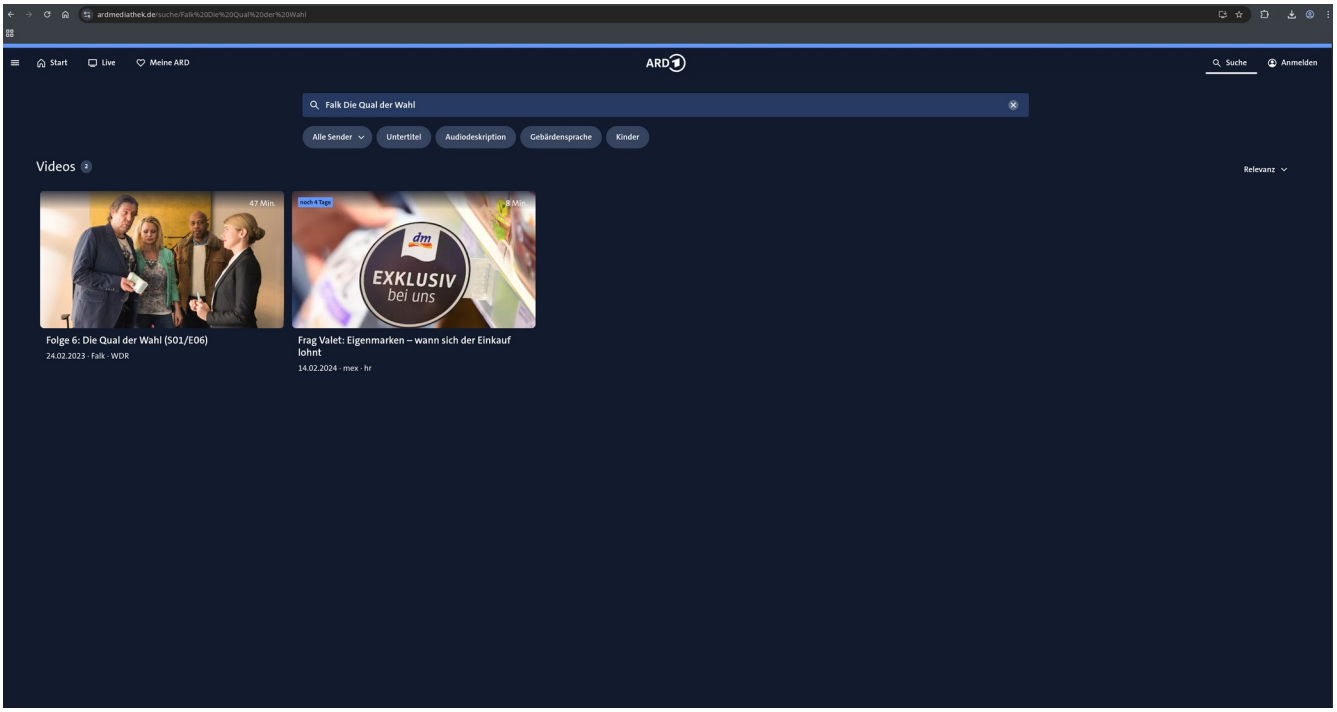
The search bar is this element:

 <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" class="b12o3cf2"><path d="m20.72 19.34-5.52-5.59a6.82 6.82 0 0 0 1.24-3.93c0-3.76-3.01-6.81-6.72-6.81S3 6.05 3 9.81s3.01 6.81 6.72 6.81c1.57 0 3.01-.55 4.16-1.47l5.49 5.56c.19.19.43.29.68.29s.49-.1.68-.29c.38-.38.38-1 0-1.38zM4.92 9.81c0-2.68 2.15-4.86 4.8-4.86s4.8 2.18 4.8 4.86-2.15 4.86-4.8 4.86-4.8-2.18-4.8-4.86"></path></svg> Suche

But it can be searched with just the attribute aria-label=”Suche” for more simplicity

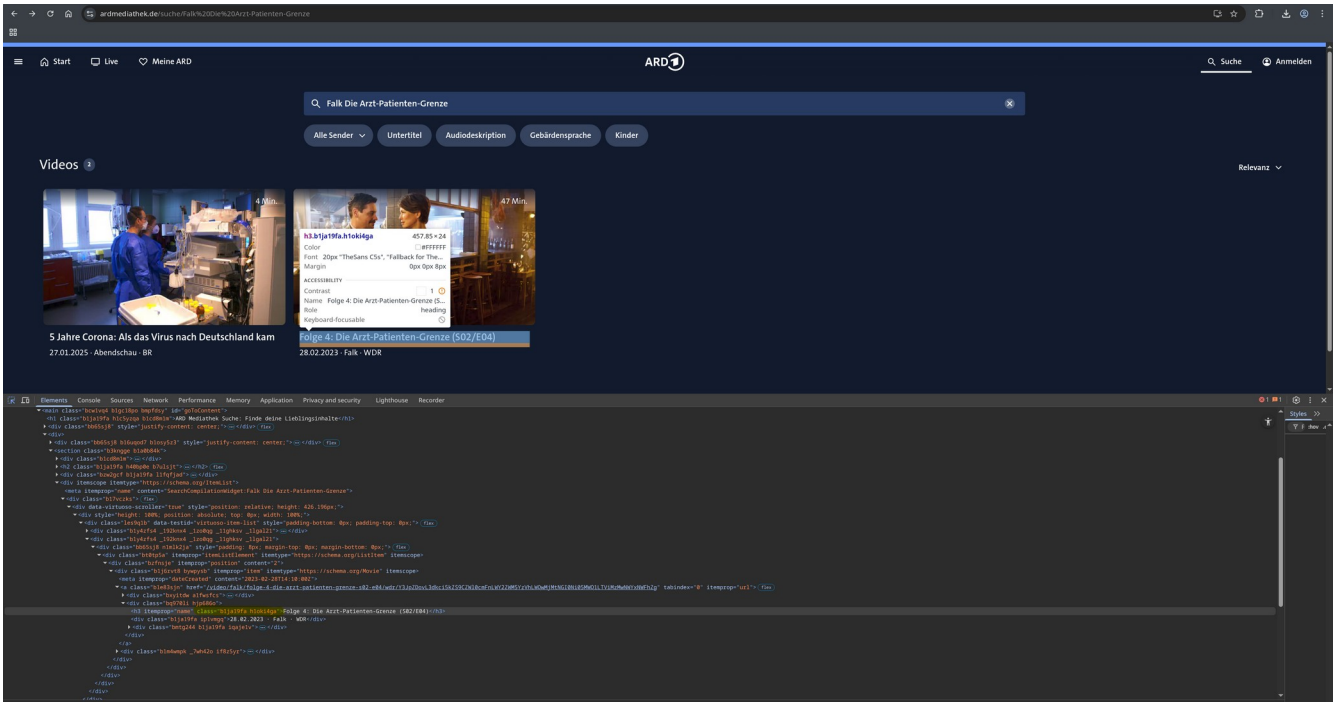


when clicking on the element, it will redirect to this url https://www.ardmediathek.de/suche/ where an input field is working so you just need to type the series you want to watch



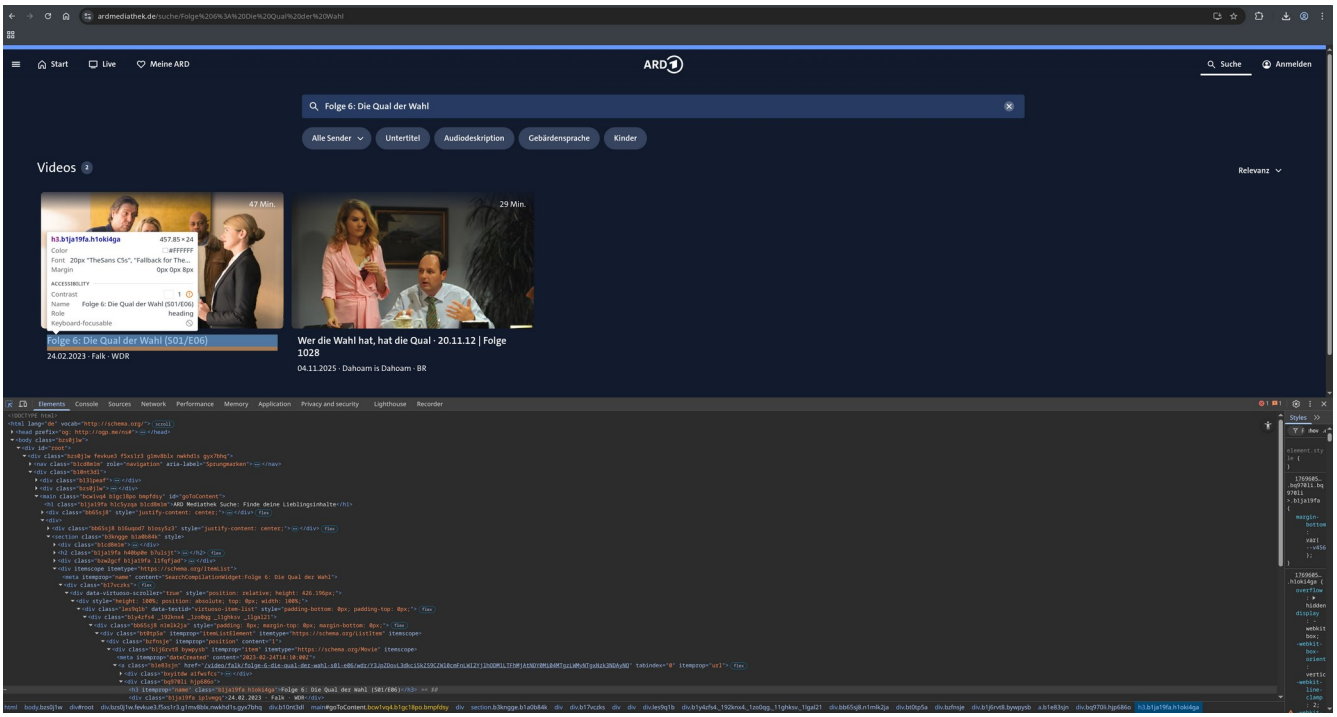
so we type the name of the tv show + name of episode; the url will also have the query params to search for it. 99% of the cases it is the first element. I am saying 99% because there is also this episode

when the episode is the second in the search list

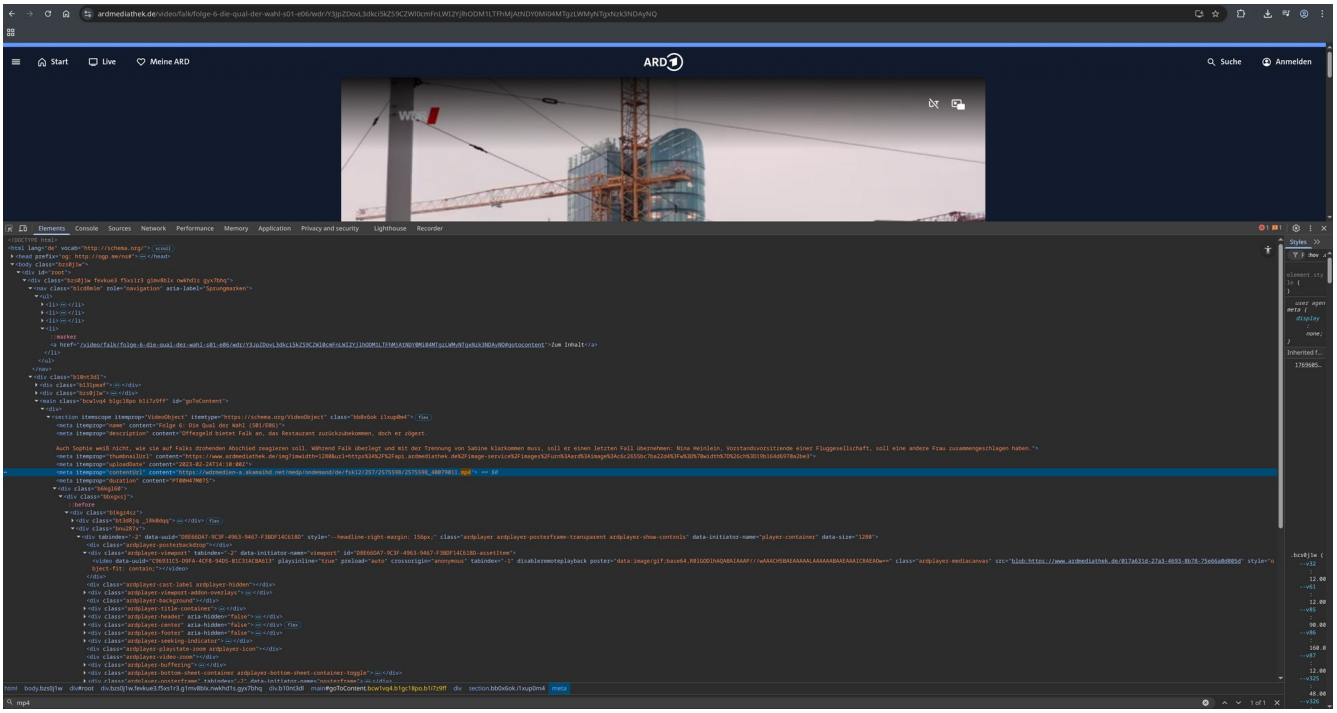


both have the same class “b1ja19fa h1oki4ga” .

ok. so back to first example:

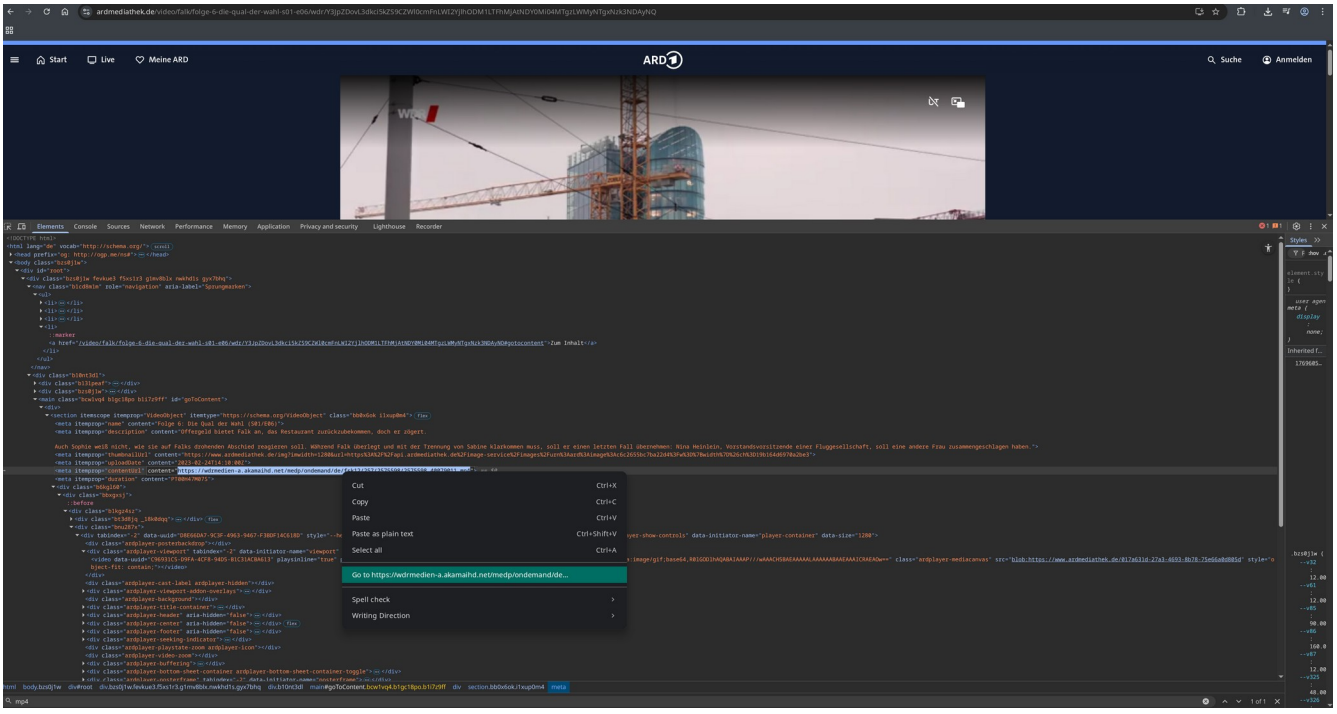


again we can see the same class “b1ja19fa h1oki4ga” . Now if we click on this element, we will be redirected to the video.

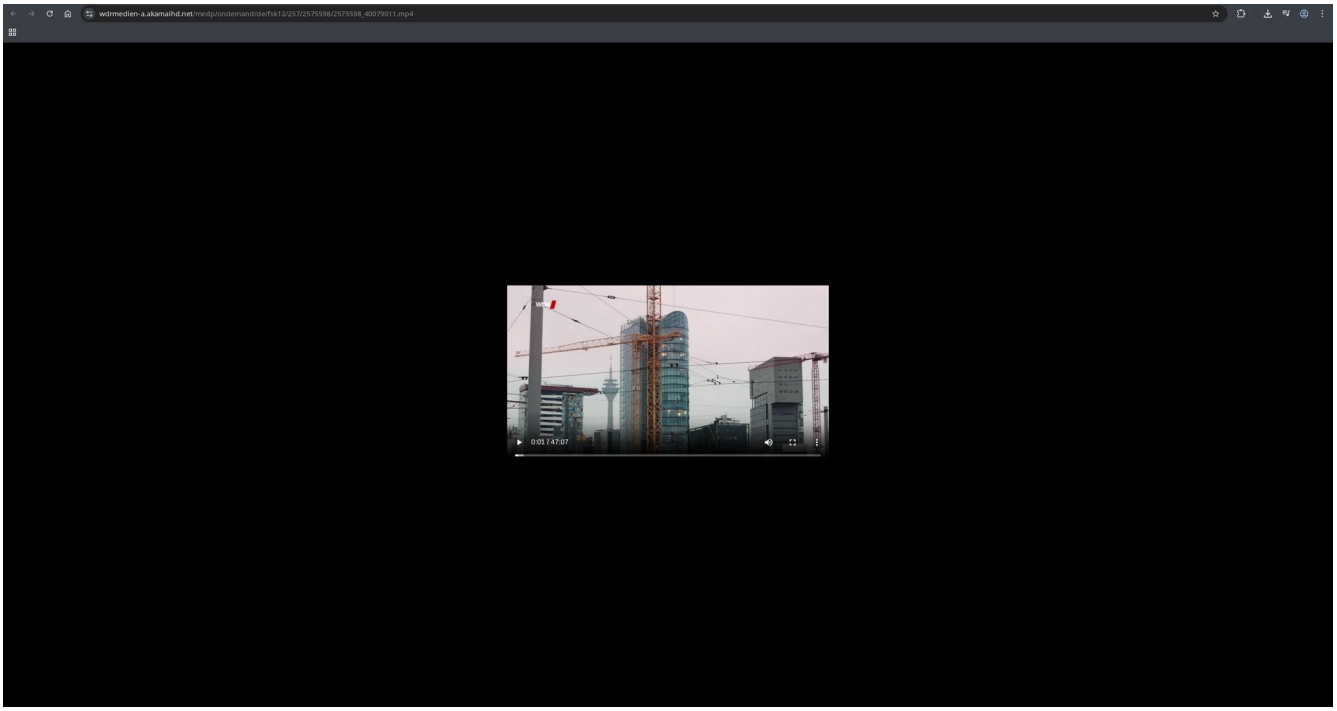


If we click inspect and search for mp4 we see this interesting content with a mp4 extension

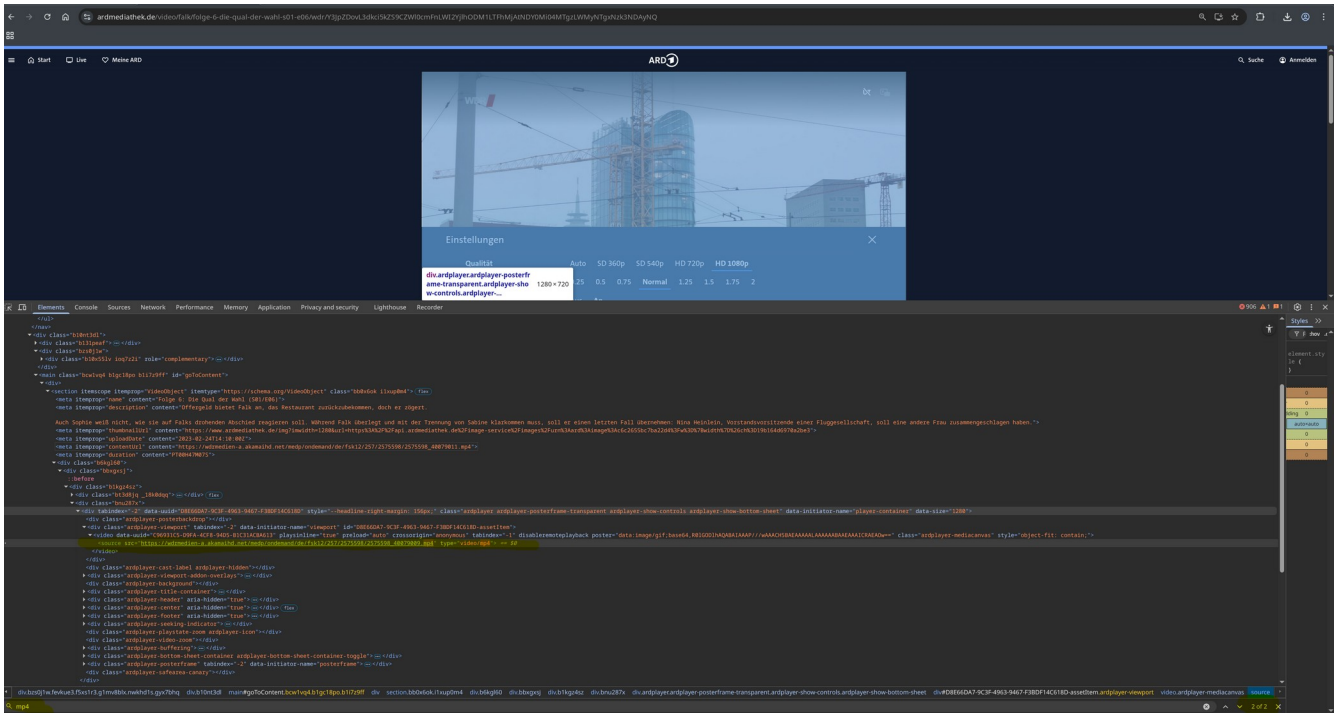
<meta itemprop="contentUrl" content="https://wdrmedien-a.akamaihd.net/medp/ondemand/de/fsk12/257/2575598/2575598_40079011.mp4">



we can even go to this link, and we'll be redirected to:

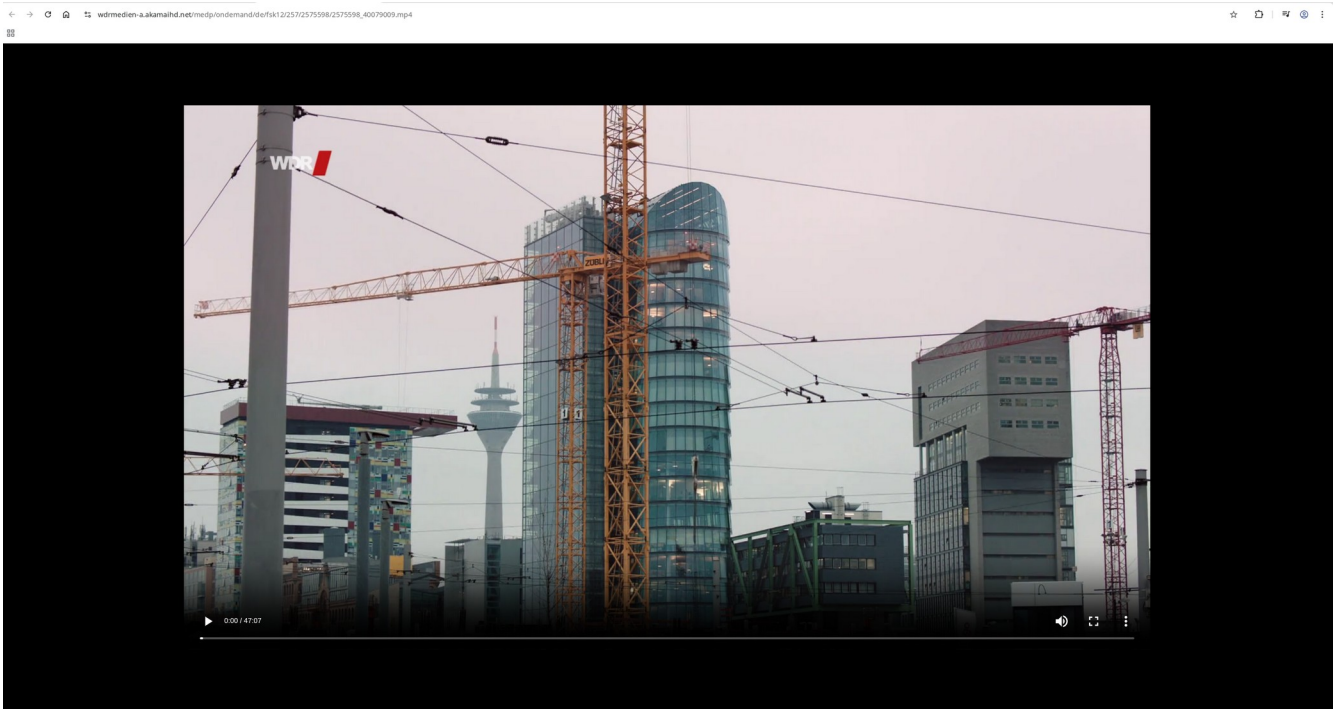


to our video. A video with bad quality though. So what's the problem? Turns out, when we go back, if we didn't set ourselves Settings → HD Quality 1080p, it will be stuck on Auto or last video quality. And here's also an interesting part, it appears that when inspecting again on Element from browser after we set the video quality from Settings, we will see two mp4's!



The second video is the quality that we selected from Settings

and if we open the link of the second video, now we get the best quality

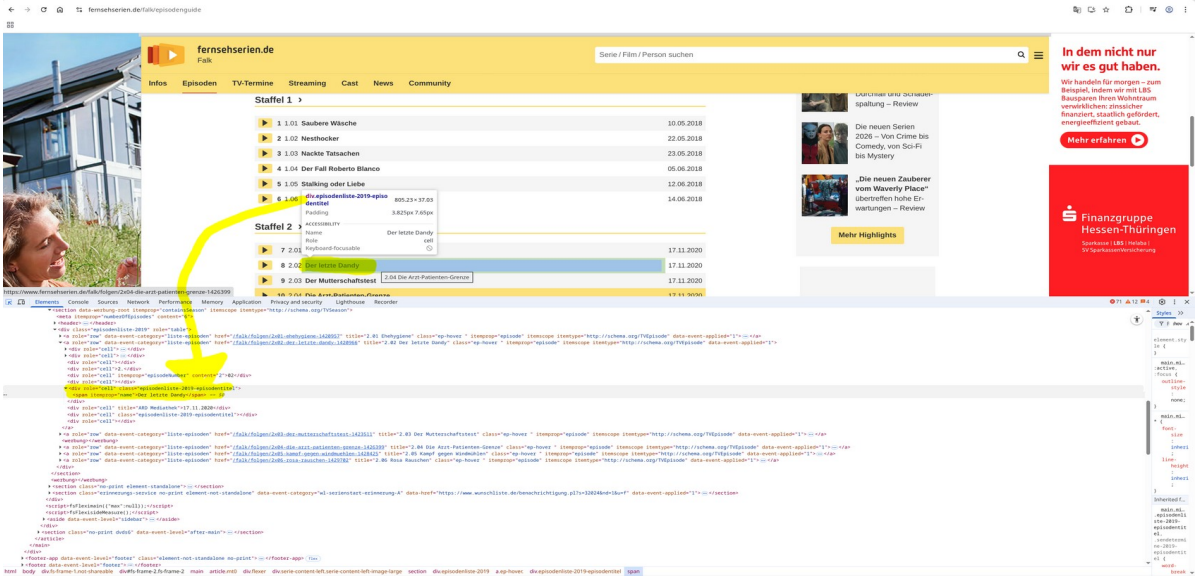


This is what we want to code. We want to webscrap until we find this link and download the videos in a folder.

But it won’t work as stated before without the vpn extension on the browser. This is why we use Puppeteer, due to Geolocation and the fact that we can’t use a simple fetch api through Node JS in this context.

Code explanation:

test.js:



We fetch this URL to get all the episode names. Each episode name is inside a span element that is inside a div with class name .episodenliste-2019-episodentitel. Then we export the function to our second script.

```
const { JSDOM } = require("jsdom");
const { pipeline } = require("node:stream/promises");
const fs = require("node:fs");

const infoURL = "https://www.fernsehserien.de/falk/episodenguide";

async function getAllEpisodeTitles() {
  try {
    const response = await fetch(infoURL);
    if (response.ok) {
      const data = await response.text();
      const dom = new JSDOM(data);
      const document = dom.window.document;

      const divs = document.querySelectorAll(
        ".episodenliste-2019-episodentitel",
      );

      const episodeTitles = [...divs]
        .map((div) => div.querySelector("span")?.textContent.trim())
        .filter(Boolean);

      console.log(episodeTitles);
      return episodeTitles;
    } else {
      console.log("Failed to find titles");
      return [];
    }
  } catch (err) {
    console.error(err);
    return [];
  }
}


module.exports = { getAllEpisodeTitles };
```

puppet1.js

We export the function from the first script into this script, as well puppeteer module, fs module, path module and pipeline module

```
const downloadDir = path.join(__dirname, "downloadDir");
```
is for telling the script where we want to put the downloaded videos. In this case in **downloadDir**

And now the big **(async ()**

1) First we retrieve the episode names by using the function exported from the frist script

```
const episodes = await getAllEpisodeTitles();
```

2)We explicitly say to puppeteer that were are using this type of browser:

```
const chromeBrowser = {
browserURL: "http://127.0.0.1:9222",
defaultViewport: null,
protocolTimeout: timeout,
headless: false,
};

const browser = await puppeteer.connect(chromeBrowser);
```
3) inside const download there is an async function that take **episode** as param

before downloading videos we check if there are already downloaded videos so that they are not download again. If they do not exists, return

```
const filePath = path.join(downloadDir, episodeFile);
console.log(
filePath,
fs.existsSync(filePath) || fs.existsSync(`${filePath}.crdownload`),
);
if (fs.existsSync(filePath) || fs.existsSync(`${filePath}.crdownload`))
return;
```

4) we open the page using puppeteer and create a client **CDPSSession** in order to let browser download the video in downloadDir

```
const page = await browser.newPage();
page.setDefaultTimeout(timeout);

const client = await page.target().createCDPSession();

await client.send("Page.setDownloadBehavior", {
behavior: "allow",
downloadPath: downloadDir,
});
```

5) This is the automated Manual walktrhough explained earlier

```
await page.goto("https://www.ardmediathek.de/");

await page.setViewport({ width: 1024, height: 1024 });

const findSuche = await page.locator('[aria-label="Suche"]').waitHandle();

await findSuche?.click();

const sucheInput = await page.locator("#SearchInputWidget").waitHandle();
await sucheInput?.click();

await page.keyboard.type(`Falk ${episode}`, { delay: 50 });

await page.keyboard.press("Enter", { delay: 50 });

const titleVideo = await page.locator(".b1ja19fa.h1oki4ga").waitHandle();
await titleVideo?.click();

await delayExecutionFor(1000);
const videoSettings = await page
  .locator(
    'button[class="ardplayer-button-settings ardplayer-icon ardplayer-icon-settings ardplayer-icon-settings-hd"]' +
      '[data-initiator-name="addon-button(SettingsSheetAddon:settings)"]' +
      '[tabindex="0"]' +
      '[title="Einstellungen an / aus"]' +
      '[data-display-title="Einstellungen"]' +
      '[aria-pressed="false"]' +
      '[aria-label="Einstellungen an / aus"]',
  )
  .waitHandle();

await delayExecutionFor(1000);
await videoSettings?.click();

await page
  .locator(
    "span" +
      '[tabindex="0"]' +
      '[role="option"]' +
      '[data-index="4"]' +
      '[data-initiator-name="select(Qualität)=>HD 1080p"]',
    //'[aria-selected="false"]',
  )
  ?.click();

const videoSelector = await page
  .locator('source[type="video/mp4"]')
  .waitHandle();

const videoSRC = await videoSelector?.evaluate((el) => el.src);
```

small time delays were added in order for the page to have time to upload;

6) In const **videoSRC** is the URL with the **mp4** file with the best quality, but we also need to download the video, for that we do this

```
await page.evaluate(
(url, episodeFile) => {
const a = document.createElement("a");
a.href = url;
a.download = episodeFile;
document.body.appendChild(a);
a.click();
a.remove();
},
videoSRC,
episodeFile,
);
```

7) The video is being downloaded and now we just close the page so that browser won't crash when going to the next episode

```
await page.close();
console.log(`Completed: ${episode}`);
};
```

8) And we do a for loop

```
//await download(episodes[0]);
for (const episode of episodes) {
await download(episode);
}
})();
```

Thank you for coming to coming to my tedtalk
100% no AI written