BY SAM AGNEW • 2020-04-08

TWITTER FACEBOOK LINKEDIN

Web Scraping and Parsing HTML in Node.js with jsdom

The internet has a wide variety of information for human consumption. But this data is often difficult to access programmatically if it doesn't come in the form of a dedicated REST API. With

Node.js tools like jsdom, you can scrape and parse this data directly from web pages to use for your projects and applications.

Let's use the example of needing MIDI data to train a neural network that can generate classic Nintendo-sounding music. In order to do this, we'll need a set of MIDI music from old Nintendo games. Using jsdom we can scrape this data from the Video Game Music Archive.

Getting started and setting up dependencies

Before moving on, you will need to make sure you have an up to date version of Node.js and npm installed.

Navigate to the directory where you want this code to live and run the following command in your terminal to create a package for this project:

```
npm init --yes
```

The `--yes` argument runs through all of the prompts that you would otherwise have to fill out or skip. Now we have a package.json for our app.

For making HTTP requests to get data from the web page we will use the Got library, and for parsing through the HTML we'll use Cheerio.

Run the following command in your terminal to install these libraries:

```
npm install got@10.4.0 jsdom@16.2.2
```

jsdom is a pure-JavaScript implementation of many web standards, making it a familiar tool to use for lots of JavaScript developers. Let's dive into how to use it.

Using Got to retrieve data to use with jsdom

First let's write some code to grab the HTML from the web page, and look at how we can start parsing through it. The following code will send a `GET` request to the web page we want, and will create a jsdom object with the HTML from that page, which we'll name `dom` :

```javascript
const fs = require('fs');
const got = require('got');
const jsdom = require("jsdom");
const { JSDOM } = jsdom;

const vgmUrl= 'https://www.vgmusic.com/music/console/nintendo/nes';

got(vgmUrl).then(response => {
  const dom = new JSDOM(response.body);
  console.log(dom.window.document.querySelector('title').textContent);
}).catch(err => {
  console.log(err);
});
```

When you pass the `JSDOM` constructor a string, you will get back a JSDOM object, from which you can access a number of usable properties such as `window`. As seen in this code, you can navigate through the HTML and retrieve DOM elements for the data you want using a query selector.

For example, `querySelector('title').textContent` will get you the text inside of the `<title>` tag on the page. If you save this code to a file named `index.js` and run it with the command `node index.js`, it will log the title of the web page to the console.

Using CSS Selectors with jsdom

If you want to get more specific in your query, there are a variety of selectors you can use to parse through the HTML. Two of the most common ones are to search for elements by class or ID. If you wanted to get a div with the ID of "menu" you would use `querySelectorAll('#menu')` and if you wanted all of the header columns in the table of VGM MIDIs, you'd do `querySelectorAll('td.header')`

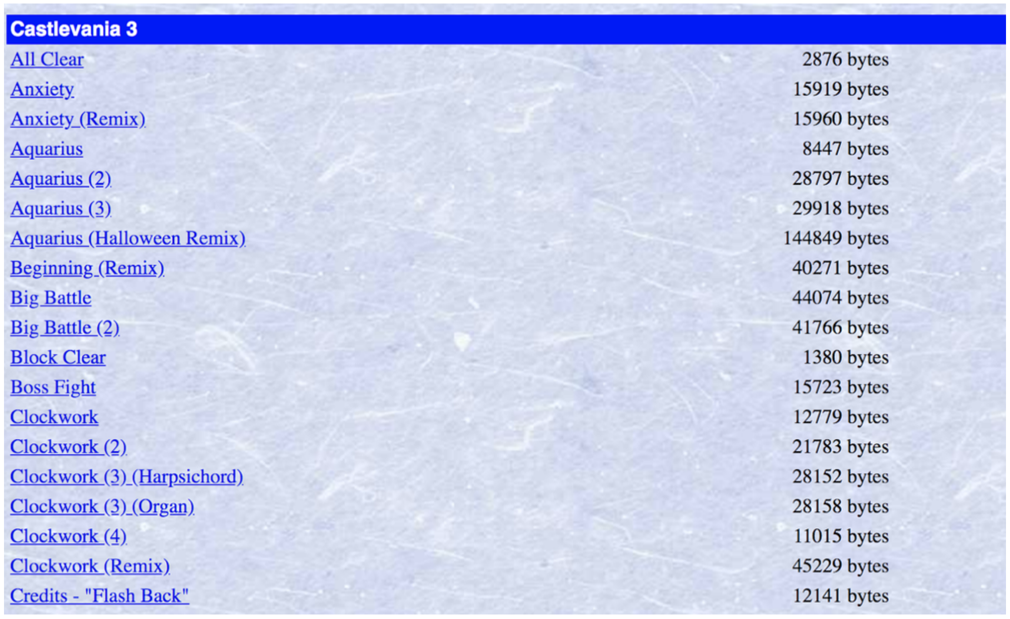
What we want on this page are the hyperlinks to all of the MIDI files we need to download. We can start by getting every link on the page using `querySelectorAll('a')`. Add the following to your code in `index.js`:

```javascript
got(vgmUrl).then(response => {
  const dom = new JSDOM(response.body);
    dom.window.document.querySelectorAll('a').forEach(link => {
    console.log(link.href);
  });
}).catch(err => {
  console.log(err);
});
```

This code logs the URL of every link on the page. We're able to look through all elements from a given selector using the `forEach` function. Iterating through every link on the page is great, but we're going to need to get a little more specific than that if we want to download all of the MIDI files.

Filtering through HTML elements

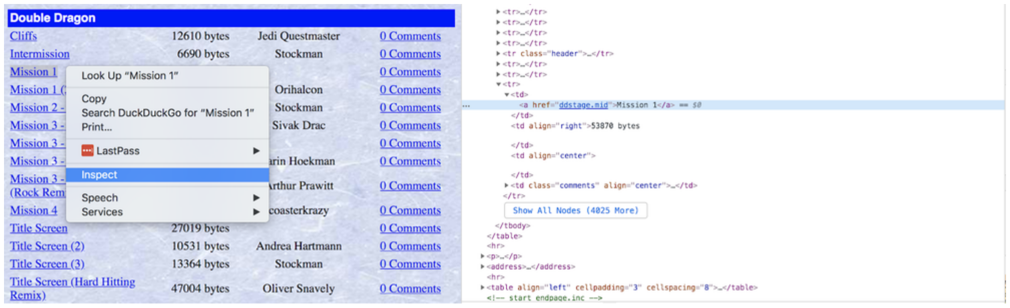
Before writing more code to parse the content that we want, let's first take a look at the HTML that's rendered by the browser. Every web page is different, and sometimes getting the right data out of them requires a bit of creativity, pattern recognition, and experimentation.



Castlevania 3	
All Clear	2876 bytes
Anxiety	15919 bytes
Anxiety (Remix)	15960 bytes
Aquarius	8447 bytes
Aquarius (2)	28797 bytes
Aquarius (3)	29918 bytes
Aquarius (Halloween Remix)	144849 bytes
Beginning (Remix)	40271 bytes
Big Battle	44074 bytes
Big Battle (2)	41766 bytes
Block Clear	1380 bytes
Boss Fight	15723 bytes
Clockwork	12779 bytes
Clockwork (2)	21783 bytes
Clockwork (3) (Harpsichord)	28152 bytes
Clockwork (3) (Organ)	28158 bytes
Clockwork (4)	11015 bytes
Clockwork (Remix)	45229 bytes
Credits - "Flash Back"	12141 bytes

Our goal is to download a bunch of MIDI files, but there are a lot of duplicate tracks on this webpage, as well as remixes of songs. We only want one of each song, and because our ultimate goal is to use this data to train a neural network to generate accurate Nintendo music, we won't want to train it on user-created remixes.

When you're writing code to parse through a web page, it's usually helpful to use the developer tools available to you in most modern browsers. If you right-click on the element you're interested in, you can inspect the HTML behind that element to get more insight.



Double Dragon			
Cliffs	12610 bytes	Jedi Questmaster	0 Comments
Intermission	6690 bytes	Stockman	0 Comments
Mission 1			0 Comments
Mission 1 (		Orihalcon	0 Comments
Mission 2 -		Stockman	0 Comments
Mission 3 -		Sivak Drac	0 Comments
Mission 3 -			0 Comments
Mission 3 -		arin Hoekman	0 Comments
Mission 3 - (Rock Rem		Arthur Prawitt	0 Comments
Mission 4		coasterkrazy	0 Comments
Title Screen	27019 bytes		0 Comments
Title Screen (2)	10531 bytes	Andrea Hartmann	0 Comments
Title Screen (3)	13364 bytes	Stockman	0 Comments
Title Screen (Hard Hitting Remix)	47004 bytes	Oliver Snavely	0 Comments

```
      ▸<tr>…</tr>
      ▸<tr>…</tr>
      ▸<tr>…</tr>
      ▸<tr>…</tr>
      ▸<tr class="header">…</tr>
      ▸<tr>…</tr>
      ▸<tr>…</tr>
      ▾<tr>
        ▾<td>
...        <a href="ddstage.mid">Mission 1</a> == $0
         </td>
         <td align="right">53870 bytes

         </td>
         <td align="center">

         </td>
        ▸<td class="comments" align="center">…</td>
       </tr>
        Show All Nodes (4025 More)
      </tbody>
    </table>
    <hr>
   ▸<p>…</p>
   ▸<address>…</address>
    <hr>
   ▸<table align="left" cellpadding="3" cellspacing="8">…</table>
    <!-- start endpage.inc -->
```

You can write filter functions to fine-tune which data you want from your selectors. These are functions which loop through all elements for a given selector and return true or false based on whether they should be included in the set or not.

If you looked through the data that was logged in the previous step, you might have noticed that there are quite a few links on the page that have no `href` attribute, and therefore lead nowhere. We can be sure those are not the MIDIs we are looking for, so let's write a short function to filter those out as well as elements which do contain a `href` element that leads to a `.mid` file:

```javascript
const isMidi = (link) => {
  // Return false if there is no href attribute.
  if(typeof link.href === 'undefined') { return false }

  return link.href.includes('.mid');
};
```

Now we have the problem of not wanting to download duplicates or user generated remixes. For this we can use regular expressions to make sure we are only getting links whose text has no parentheses, as only the duplicates and remixes contain parentheses:

```javascript
const noParens = (link) => {
  // Regular expression to determine if the text has parentheses.
  const parensRegex = /^((?!\().)*$/;
  return parensRegex.test(link.textContent);
};
```

Try adding these to your code in `index.js` by creating an array out of the collection of HTML Element Nodes that are returned from `querySelectorAll` and applying our filter functions to it:

```
got(vgmUrl).then(response => {
  const dom = new JSDOM(response.body);

  // Create an Array out of the HTML Elements for filtering using sprea
  const nodeList = [...dom.window.document.querySelectorAll('a')];

  nodeList.filter(isMidi).filter(noParens).forEach(link => {
    console.log(link.href);
  });
}).catch(err => {
  console.log(err);
});
```

Run this code again and it should only be printing `.mid` files, without duplicates of any particular song.

Downloading the MIDI files we want from the webpage

Now that we have working code to iterate through every MIDI file that we want, we have to write code to download all of them.

In the callback function for looping through all of the MIDI links, add this code to stream the MIDI download into a local file, complete with error checking:

```
  nodeList.filter(isMidi).filter(noParens).forEach(link => {
    const fileName = link.href;
    got.stream(`${vgmUrl}/${fileName}`)
      .on('error', err => { console.log(err); console.log(`Error on ${v
      .pipe(fs.createWriteStream(`MIDIs/${fileName}`))
      .on('error', err => { console.log(err); console.log(`Error on ${v
      .on('finish', () => console.log(`Downloaded: ${fileName}`));
  });
```

Run this code from a directory where you want to save all of the MIDI files, and watch your terminal screen display all 2230 MIDI files that you downloaded (at the time of writing this). With that, we should be finished scraping all of the MIDI files we need.

```
Downloaded: Levels 6 and 9
Downloaded: Cave
Downloaded: Great Palace Boss
Downloaded: House
Downloaded: Overworld
Downloaded: Palace
Downloaded: Palace Boss
Downloaded: Princess Zelda Awakes
Downloaded: Title Screen
Downloaded: Town
Downloaded: Whistle
```

Go through and listen to them and enjoy some Nintendo music!

The vast expanse of the World Wide Web

Now that you can programmatically grab things from web pages, you have access to a huge source of data for whatever your projects need. One thing to keep in mind is that changes to a web page's HTML might break your code, so make sure to keep everything up to date if you're building applications on top of this. You might want to also try comparing the functionality of the jsdom library with other solutions by following tutorials for web scraping using Cheerio and headless browser scripting using Puppeteer or a similar library called Playwright.

If you're looking for something to do with the data you just grabbed from the Video Game Music Archive, you can try using Python libraries like Magenta to train a neural network with it.

I'm looking forward to seeing what you build. Feel free to reach out and share your experiences or ask any questions.

- Email: sagnew@twilio.com
- Twitter: @Sagnewshreds
- Github: Sagnew
- Twitch (streaming live code): Sagnewshreds

RATE THIS POST ☆☆☆☆☆

AUTHORS Sam Agnew

Search

Build the future of communications. Start today with Twilio's APIs and services.

START BUILDING FOR FREE

POSTS BY STACK

JAVA PHP RUBY .NET PYTHON SWIFT ARDUINO JAVASCRIPT

POSTS BY PRODUCT

EMAIL SMS VOICE TWILIO CLIENT MMS VIDEO CONVERSATIONS TASK ROUTER VERIFY FLEX SIP IOT STUDIO

CATEGORIES

Code, Tutorials and Hacks

Customer Highlights

Developers Drawing The Owl

Life Inside: We Build At Twilio

News

Stories From The Road

LANGUAGES

JAPANESE GERMAN SPANISH PORTUGUESE FRENCH

TWITTER FACEBOOK

Developer stories to your inbox.

Subscribe to the Developer Digest, a monthly dose of all things code.

Enter your email...

You may unsubscribe at any time using the unsubscribe link in the digest email. See our privacy policy for more information.

NEW!

Tutorials

Sample applications that cover common use cases in a variety of languages. Download, test drive, and tweak them yourself.

Get started

SIGN UP AND START BUILDING

Not ready yet? Talk to an expert.

ABOUT
LEGAL