Building JSON from the content of a page

As part of a larger project I had to figure out how to use node to scrape the content of a page to build a JSON file from the components using Node. Rather than scrape the page I discovered a REST API end point that will provide search results as JSON. This is the function I'm using to fetch the JSON, extract its components and build the new JSON object that I'll pass to other functions.

It is not pretty... I'm the first one to admit that. But it does the work until I find something better :)

Problem Statement

As part of building an action for Google assistant that will add voice to the search results for my blog I came up with the following issue:

> How do I provide a JSON payload for assistant to use if the Wordpress generated by the WordPress REST API provides way more information than we need

Different solutions

One quick solution is to scrape the page using tools like Cheerio but it still only provides HTML content, not the JSON I need. It is possible but the process is cumbersome and we need to change the code every time we change the page.

I could also use the JSON I received from Wordpress as is but it makes for a large download for only using a fraction of the data in the resulting prodcut.

I decided to go with node-fetch for my solution. Node-fetch is a Node implementation of the WHATWG fetch standard and allows me to do both promise and async/await code. I went with the later option to make myself be comfortable with async/await.

The code below takes a single parameter representing the query that we want to search for. With that wuery the function will:

1. Create an empty array to store the data we collect
2. Create an async function
3. Replace all the spaces in the query parameter with + signs
4. Await for the fetching of the encoded query. We use a template string literal to interpolate the value of the query
5. Convert the response to JSON
6. Using a for loop
7. Create an item array passing title, link and excerpt. Since the excerpt begins with a HTML tag we strip it by slicing the element starting at position 3 (0-based) and going for 150 characters
8. Run the element through JSON.stringify and push the result into the `jsonData` array that we created in step 1
9. Log any errors to the console
10. Execute the function with a parameter to search. We make sure that it has two words to exercise all aspects of the function

```
const fetch = require('node-fetch');

const jsonData = []; // 1
const buildJSON = async function(query) { // 2
  try {
    encodedQuery = query.replace(new RegExp('\\+', 'g'), ' '); '\\+'// 3
    const response = await fetch(`https://publishing-project.rivendellweb
    const json = await response.json(); // 5
    // Do the work here instead of the console.log
    for (let i = 0; i < json.length; i++) { // 6
      const item = { // 7erpt.rendered.slice(3, 150),
      };
      jsonData.push(JSON.stringify(item)); // 8
    }
  } catch (err) { // 9
    console.log('There\'s been an error getting the data', 'There\'s been
    }
  } catch (err) { // 9
    console.log('There\'s been an error getting the data', err);
  };
};

buildJSON('lazy loading'); // 10
```

Conclusion

While this function was first created to work converting one JSON file into another we can replace the for loop with whatever instructions that we need to accomplish our goals.