fetch data from an external API

Step 2

The *Fetch API* is a built-in JavaScript interface to make network requests to a server. It has a fetch() method you can use to make GET, POST, PUT, or PATCH requests. In this project, you'll make a GET request to a URL for a JSON file with information about authors on freeCodeCamp News.

Here is how you can make a GET request with the fetch() method:

Example Code

```
fetch("url-goes-here")
```

Make a GET request to this URL: "https://cdn.freecodecamp.org/curriculum/news-author-page/authors.json". Don't terminate your code with a semicolon yet.

Step 3

The fetch() method returns a Promise, which is a placeholder object that will either be fulfilled if your request is successful, or rejected if your request is unsuccessful.

If the Promise is fulfilled, it resolves to a Response object, and you can use the .then() method to access the Response.

Here's how you can chain .then() to the fetch() method:

Example Code

```
fetch("sample-url-goes-here")
  .then((res) => res)
```

Chain the .then() method to your fetch call. Inside the .then() method, add a callback function with res as a parameter, then log the res to the console so you can see

the Response object. Open your browser console and expand the Response object to see what it contains.

Again, don't terminate the code with a semicolon yet.

Step 4

The data you get from a GET request is not usable at first. To make the data usable, you can use the .json() method on the Response object to parse it into JSON. If you expand the Prototype of the Response object in the browser console, you will see the .json() method there.

Remove console.log(res) and implicitly return res.json() instead.

Step 6

The .catch() method is another asynchronous JavaScript method you can use to handle errors. This is useful in case the Promise gets rejected.

Chain .catch() to the last .then(). Pass in a callback function with err as the parameter. Inside the callback, use console.error() to log possible errors to the console with the text There was an error: ${err}. Since you're using err in the text, don't forget to use a template literal string with backticks (``) instead of single or double quotes.

Note: Now you can terminate your code with a semicolon. You couldn't do that in the previous steps because you'll signal to JavaScript to stop parsing your code, which will affect the fetch() syntax.

Step 7

Now that you have the data you want, you can use it to populate the UI. But the fetched data contains an array of 26 authors, and if you add them all to the page at the same time, it could lead to poor performance.

Instead, you should add 8 authors at a time, and have a button to add 8 more until there's no more data to display.

Use `let` to create 2 variables named `startingIndex` and `endingIndex`, and assign them the number values `0` and `8`, respectively. Also, create an `authorDataArr` variable with `let` and set it to an empty array.

Step 8

Now you'll create a function to populate the UI with the author data. You will call this function inside the second `.then()` method.

Create an empty arrow function named `displayAuthors` that takes `authors` as a parameter.

Step 16

Now it's time to call the `displayAuthors` function. But again, you don't want to populate the page with all the authors at once. Instead, you can extract a portion of the authors with the `startingIndex` and `endingIndex` variables. The best method to do this is the `.slice()` array method.

First, remove the console log statement showing `authorDataArr`. Then, call the `displayAuthors` function with the `authorDataArr` array and `.slice()`. Use the `startingIndex` variable for the starting point and the `endingIndex` variable for the ending point.

Step 24

Your fCC Authors Page is now complete. But you could improve on a few things.

First, if you click the `Load More Authors` button a couple of times, you'll see that it won't add more authors to the page. That's because you've reached the end of the authors list. For a better user experience, you should make it clear when there's no

more data to display by disabling the button and changing its text. An `if` statement is the perfect tool for this.

Inside the `fetchMoreAuthors` function, write an `if` statement and set the condition to `authorDataArr.length <= endingIndex` – meaning there's no more data to load.