LEARN FETCH AND PROMISES BY BUILDING AN fCC AUTHORS PAGE

Introduction:

One common aspect of web development is learning how to fetch data from an external API, then work with asynchronous JavaScript.

This freeCodeCamp authors page project will show you how to use the fetch method, then dynamically update the DOM to display the fetched data.

This project will also teach you how to paginate your data so you can load results in batches.

Step 1:

All the HTML and CSS for this project has been provided for you. You can take a look at the two files to familiarize yourself with them.

Start by getting the `#author-container` and `#load-more-btn` elements with the `.getElementById()` method. Assign them to the variables `authorContainer` and `loadMoreBtn`, respectively.

The variables will not change, so use `const` to declare them.

Step 2:

In this project we want data about the authors on freeCodeCamp News. If you want data from an online source, you need use an `API` (Application Programming Interface). An `API` lets people from outside of an organization retrieve its internal data.

There is a method called `fetch` that allows code to receive data from an `API` by sending a `GET` request.

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 5:

In order to start working with the data, you will need to use another `.then()` method.

Chain another `.then()` to the existing `.then()` method. This time, pass in `data` as the parameter for the callback function. For the callback, use a curly brace because you will have more than one expression. Within your callback function, log `data` to the console to see what it looks like.

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 9:

Inside your `displayAuthors` function, chain `.forEach()` to `authors`.

Step 10:

Pass an empty callback function to the `.forEach()` method. For the first parameter of the callback, destructure the `author`, `image`, `url`, and `bio` items.

For the second parameter, pass in `index`. This will represent the position of each author, and will be useful for pagination later.

Step 11:

Now it's time to start building the HTML for the page with your destructured data. You can do this with a combination of the compound assignment operator (`+=`) and the `innerHTML` property.

Inside your callback function, use the compound assignment operator to append an empty template literal to the `innerHTML` of `authorContainer`.

Step 12:

Inside the template literal, create a `div` element with the `id` set to the `index` from the `.forEach()` array method. Remember to use string interpolation to do this.

Also, add a `class` of `"user-card"` to the `div`.

Step 13:

Now you need to show some information about the author. First, show the author's name.

Create an `h2` tag with the `class "author-name"`. Then, interpolate `author` inside the `h2` tag. This is the author's name.

Step 14:

To see the authors' names on the page, you need to call the `displayAuthors` function inside the second `.then()` method. But before that, you need to assign the author data to the empty `authorDataArr` array.

First, remove your `console.log()` statement. Then, assign `data` to the `authorDataArr` variable.

Step 15:

Remember that `range()` returns an array, so you can chain array methods directly to the function call.

Call `range()` with `1` and `99` as the arguments, and chain the `.forEach()` method. Pass the `.forEach()` method an empty callback which takes `number` as the 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 17:

Now create an image tag and give it the class `"user-img"`. Use string interpolation to set the `src` attribute to `image` you destructured earlier. Set the `alt` attribute to `author` followed by the text `"avatar"`. Make sure there is a space between the `author` variable and the word `"avatar"`, for example, `"Quincy Larson avatar"`.

Step 18:

The next thing you'll show are biographical details about the author. You can do this with `bio` that you destructured earlier.

Add a paragraph element with the `class "bio"`, then interpolate `bio` inside the paragraph element.

Step 19:

Next, add a link to the author's page on freeCodeCamp News.

Add an anchor element with the `class "author-link"`, interpolate `url` as the value for the `href` attribute, and set `target` to `"_blank"`. For the text of the anchor element, interpolate `author` followed by the text `"'s author page"`. For example, `"Quincy Larson's author page"`.

Step 20:

Now you have everything you want to include in the UI. The next step is to make the `Load More Authors` button fetch more authors whenever it's clicked. You can do this by adding a `click` event to the button and carefully incrementing the `startingIndex` and `endingIndex` variables.

Create a `fetchMoreAuthors` function with the arrow function syntax. Don't put anything in it yet. Make sure you use curly braces because you'll have more than one expression inside the function.

Step 21:

Inside the `fetchMoreAuthors` function, set the `startingIndex` and `endingIndex` variables to `+= 8` each.

Step 22:

Now call the displayAuthors function with a portion of the author data just like you did before.

If you click the Load More Authors button after calling the function, it won't work. That's because you still have to add the click event listener to the button. You'll do that next.

Step 23:

Remember that in step 1 you selected the Load More Authors button and assigned it to loadMoreBtn.

Use addEventListener to add a "click" event listener to loadMoreBtn. Also, pass in a reference to the fetchMoreAuthors function to run whenever the button is clicked.

After that, when you click the button you should see 8 more authors.

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.

Step 25:

If this condition is met, disable the button by setting its `disabled` property to `true`. Also, set the `textContent` of the button to `"No more data to load"`.

Step 26:

Next, there's not a lot of separation between each author's name and image, and the rest of the details on the card. A divider will give the author cards a clear visual hierarchy.

Add a `div` element above the author's bio and give it the `class` `"purple-divider"`.

Step 27:

Some of the author bios are much longer than others. To give the cards a uniform look, you can extract the first 50 characters of each one and replace the rest with an ellipsis (`"..."`). Otherwise, you can show the entire bio.

Within the paragraph element, replace `bio` with a ternary operator. For the condition, check if the length of `bio` is greater than 50. If it is, use the `.slice()` method to extract the first 50 characters of `bio` and add an ellipsis at the end. Otherwise, show the full `bio`.

Step 28:

Finally, what if there's an error and the author data fail to load? Then we need to show an error in the UI. That's exactly what the `.catch()` method is for – handling errors.

Inside the `.catch()`, remove the `console.error()` and set the `innerHTML` of the `authorContainer` to a `p` element with the `class` `"error-msg"` and text `"There was an error loading the authors"`.

Step 29:

One more thing. If you keep clicking the `Load More Authors` button until there's no more data to load and the text changes to `"No more data to load"`, the cursor value is still `pointer`. Why not change the cursor value to `not-allowed` instead?

Access the `style` property of the `Load More Authors` button and set `cursor` to `"not-allowed"`.

With that, your author page is complete!