LEARN ASYNCHRONOUS PROGRAMMING BY BUILDING AN FCC FORUM LEADERBOARD

Introduction:

JavaScript is an asynchronous programming language. And this project will help you gain proficiency in asynchronous concepts. You'll code your own freeCodeCamp forum leaderboard.

This project will cover the Fetch API, promises, Async/Await, and the try..catch statement.

Step 1:

In this project, you will build a freeCodeCamp forum leaderboard that displays the latest topics, users, and replies from the freeCodeCamp forum. The HTML and CSS have been provided for you. Feel free to explore them.

When you are ready, use `const` to declare a `forumLatest` variable and assign it the string `"https://cdn.freecodecamp.org/curriculum/forum-latest/latest.json"`.

Below that, create another `const` variable called `forumTopicUrl` and assign it the string `"https://forum.freecodecamp.org/t/"`.

Step 2:

Next, create a `const` variable called `forumCategoryUrl` and assign it the string `"https://forum.freecodecamp.org/c/"`.

Below that, create another `const` variable called `avatarUrl` and assign it the string `"https://sea1.discourse-cdn.com/freecodecamp"`.

Step 3:

Next, access the #posts-container element by using the getElementById() method. Assign it to a new constant called postsContainer.

Step 4:

To populate the forum leaderboard with data, you will need to request the data from an API. This is known as an asynchronous operation, which means that tasks execute independently of the main program flow.

You can use the async keyword to create an asynchronous function, which returns a promise.

Example Code:

```
const example = async () => {

  console.log("this is an example");

};
```

Use the async keyword to create an asynchronous arrow function called fetchData.

Step 5:

In the last project, you used the .catch() method to handle errors. Here you'll use a try...catch statement instead.

The try block is designed to handle potential errors, and the code inside the catch block will be executed in case an error occurs.

Example Code:

```
try {
  const name = "freeCodeCamp";

  name = "fCC";

} catch (err) {

  console.log(err); // TypeError: Assignment to constant variable.

}
```

Inside your fetchData function, add a try...catch statement. The catch block should have an error parameter named err.

Step 6:

n the previous project, you used fetch() with the .then() method to perform logic after the promise was resolved. Now you will use the await keyword to handle the asynchronous nature of the fetch() method.

The await keyword waits for a promise to resolve and returns the result.

Example Code:

```
const example = async () => {

  const data = await fetch("https://example.com/api");

  console.log(data);

}
```

Inside the try block, create a constant called res and assign it await fetch(). For the fetch call, pass in the forumLatest variable.

Step 7:

You want to get the response body as a JSON object. The `.json()` method of your `res` variable returns a promise, which means you will need to `await` it.

Create a constant called `data` and assign it the value of `await res.json()`.

Step 8:

To view the data results, log the `data` variable to the console inside your `try` block.

Below your `fetchData` definition, call the function and open up the console to see the results.

Step 9:

If there is an error from the `fetch` call, the `catch` block will handle it.

Inside the `catch` block, add a `console.log` to log the `err` parameter.

Also, remove your `console.log(data);` from your `try` block now that you understand what is being returned from the `fetch` call.

Step 10:

Now it is time to display the data on the page.

Start by creating an arrow function called `showLatestPosts`, which takes a single `data` parameter.

Step 11:

As you build out your `showLatestPosts()` function, you'll need to call it to see your changes.

Call the `showLatestPosts()` function at the end of your `try` block and pass in `data` for the argument.

Step 12:

Back in your `showLatestPosts()` function, use destructuring to get the `topic_list` and `users` properties from the `data` object.

Step 13:

The `topic_list` object contains a `topics` array which contains the latest topics posted to the forum.

Destructure the `topics` array from the `topic_list` object.

Step 14:

Now it is time to start populating the data inside the `postsContainer`.

Start by calling the `map()` method on your `topics` array. For the callback function, use an empty arrow function that takes `item` as a parameter.

Then assign the result of the `map()` method to `postsContainer.innerHTML`.

Step 15:

Inside the `map` method, destructure the following properties from the `item` object:

- `id`
- `title`
- `views`
- `posts_count`
- `slug`
- `posters`
- `category_id`
- `bumped_at`

Step 16:

The next step is to build out the table which will display the forum data.

Below your destructuring assignment, add a `return` keyword followed by a set of template literals. Inside those template literals, add a table row `tr` element.

Step 17:

In the preview window, you should see a column of commas. To fix this, you should chain the `join` method to your `map` method. For the separator, pass in an empty string.

Step 18:

Inside your `tr` element, add five empty `td` elements.

Step 19:

To display the topic title, add a `p` element inside the first `td` element.

Between the paragraph tags, add an embedded expression that contains the `title` variable. Then add a class called `"post-title"` inside the opening paragraph tag.

Step 20:

Keep the second `td` element empty because you will add content to it later on.

In the third `td` element, add the following embedded expression: `${posts_count - 1}`.

This will display the number of replies to the topic.

Step 21:

In the fourth `td` element, add an embedded expression that contains the `views` variable. This will display the number of views the post has.

Step 22:

To display data in the `Activity` column, you need to use the `bumped_at` property of each topic, which is a timestamp in the ISO 8601 format. You need to process this data before you can show how much time has passed since a topic had any activity.

Create a new `timeAgo` function with a `time` parameter.

Inside your `timeAgo` function, create two variables named `currentTime` and `lastPost` and set them to `new Date()` and `new Date(time)` respectively.

lastPost will be the date of the last activity on a topic, and currentTime represents the current date and time.

Step 23:

For your timeAgo function, you will want to calculate the difference between the current time and the time of the last activity on a topic. This will allow you to display how much time has passed since a topic had any activity.

Complete the timeAgo function that meets the following requirements:

- If the amount of minutes that have passed is less than 60, return the string "xm ago". x will represent the minutes.
- If the amount of hours that have passed is less than 24, return the string "xh ago". x will represent the hours.
- Otherwise, return the string "xd ago". x will represent the days.

Here are some equations that will help you calculate the time difference:

- `minutes = Math.floor((currentTime - lastPost) / 60000);`
- `hours = Math.floor((currentTime - lastPost) / 3600000);`
- ` = Math.floor((currentTime - lastPost) / 86400000);`

Step 24:

To display the time since the last post, call the timeAgo function and pass in the bumped_at variable for the argument. Place this function call inside the last td element.

Once you make those changes, scroll across the table to see the new values displayed in the Activity column.

Step 25:

You need a function to convert view counts to a more readable format. For example, if the view count is `1000`, it should display as `1k` and if the view count is `100,000` it should display as `100k`.

Create a `viewCount` function with a `views` parameter. If `views` is greater than or equal to `1000`, return a string with the `views` value divided by `1000` and the letter `k` appended to it. Make sure to round `views / 1000` down to the nearest whole number.

Otherwise, return the `views` value.

For example, if `views` is `1000` your return value should be the string `"1k"`.

Step 26:

Inside the fourth `td` element, update the current value to instead call the `viewCount` function with the `views` variable as an argument.

Step 27:

Each of the forum topics includes a category like Python or JavaScript. In the next few steps, you will build out a category object which holds all of the forum categories and `classNames` for the styling.

Start by creating a new constant called `allCategories` and assign it the value of an empty object.

Step 28:

Inside your `allCategories` object, add a new key for the number `299` with a value of an empty object.

Inside that object, add a property with a key of category and a string value of "Career Advice". Below that property, add another key called className with a string value of "career".

Step 29:

Add a new key for the number 409 with a value of an empty object.

Inside that object, add a property with a key of category and a string value of "Project Feedback".

Below that property, add another key called className with a string value of "feedback".

Step 30:

Add a new key for the number 417 with a value of an empty object.

Inside that object, add a property with a key of category and a string value of "freeCodeCamp Support".

Below that property, add another key called className with a string value of "support".

Step 31:

The rest of the allCategories object has been completed for you.

In the next few steps, you will create a function to retrieve the category name from the allCategories object.

Start by creating an arrow function named forumCategory, with id as the parameter name.

Step 32:

Inside your `forumCategory` function, create a new `let` variable named `selectedCategory` and assign it an empty object. This will be used to store the category name and class name for each category.

Step 33:

Create an `if` statement to check if the `allCategories` object has a property of `id`. Remember, you can use the `hasOwnProperty()` method for this.

Step 34:

Inside the `if` statement, destructure `className` and `category` from the `allCategories[id]` object.

Step 35:

You now need to add the `className` and `category` properties to your `selectedCategory` object.

Start by assigning the `className` variable to `selectedCategory.className`. Then assign the `category` variable to `selectedCategory.category`.

Step 36:

If the `id` is not in the `allCategories` object, you will need to display the `General` category.

Below your if statement, add an `else` clause.

Step 37:

Inside your `else` clause, assign the string `"general"` to `selectedCategory.className.`

Below that, assign the string `"General"` to `selectedCategory.category`.

Lastly, assign the number `1` to `selectedCategory.id`.

Step 38:

Every category will have a URL that points to the category on the freeCodeCamp forum.

Create a constant called `url` and assign it a template literal. Inside that template literal, place the value of `${forumCategoryUrl}${selectedCategory.className}/${id}`.

Step 39:

Create a constant called `linkText` and assign it the value of `selectedCategory.category`. This will display the name of the category in the anchor element.

Step 40:

Create a constant called `linkClass` and assign it a template literal. Inside that template literal, add the value of `category ${selectedCategory.className}`.

These class names will be used to apply styles for the anchor element.

Step 41:

Next, return an anchor element inside template literals. For the `href` attribute, assign the value of the `url` constant.

Step 42:

After the `href` attribute, set the `class` attribute to the constant named `linkClass`.

After the `class` attribute, set the `target` attribute to `"_blank"`.

Lastly, place the `linkText` constant in between the anchor tags to display the text in the link.

Step 43:

Inside the first `td` element, add an embedded expression `${}`. Inside that expression, call the `forumCategory` function with the argument of `category_id`.

Now, you should see a category displayed underneath each post topic.

Step 44:

Each forum post will include a list of user avatar images which represent all of the users participating in the conversation for that topic.

Start by creating an arrow function called `avatars`, with two parameters called `posters` and `users`.

Step 45:

The next step is to loop through the `posters` array to get all of their avatars.

Start by adding a `return` keyword followed by `posters.map()`. For the callback function, add a parameter called `poster`.

Step 46:

The next step is to find the correct user in the users array.

Start by creating a constant called user and assign it users.find(). The find method should have a callback function with a parameter of user.

Inside the callback function of the find method, implicitly return the result of checking if user.id is strictly equal to poster.user_id.

Step 47:

Next, check if the user exists. Add an if statement with user for the condition.

Step 48:

To customize the avatar's size, you can set it to a value of 30.

Start by creating a constant called avatar. Then assign it the result of using the replace method on user.avatar_template.

For the replace method, use /{size}/ for the first argument and the number 30 for the second argument.

Step 49:

Next, you will construct the userAvatarUrl.

Start by creating a constant called userAvatarUrl. Then use a ternary operator to check if avatar starts with "/user_avatar/".

If so, use the concat method to concatenate avatar to avatarUrl. Otherwise return avatar.

This will ensure the avatar URL is correctly formed whether it's a relative or absolute URL.

Step 50:

Lastly, you will need to return the image for the user avatar.

Start by adding a `return` followed by a set of template literals. Inside the template literals, add an `img` element.

Inside the `img` tag, add a `src` attribute with the value of `${userAvatarUrl}`. For the `alt` attribute, add a value of `${user.name}`.

Step 51:

At the end of your `map` method, chain the `join()` method. For the separator, pass in an empty string.

Step 52:

For the remaining steps, you will add the functionality to display the user avatars.

Inside the second `td` element, add a `div` element with a class name of `"avatar-container"`.

Step 53:

Inside the `div` element, call the `avatars` function and pass in the arguments of `posters` and `users`.

Now you should see the avatars displayed on the page.

Step 54:

Your project is almost complete. It is just missing one last piece.

Users should be able to click on any post title and be directed to the actual post on the freeCodeCamp forum.

Start by changing the existing paragraph element inside the first `td` element to be an anchor element.

Step 55:

For the opening `a` tag, set the `target` attribute to `"_blank"`. Then, set the `href` attribute to `${forumTopicUrl}${slug}/${id}`.

And with those changes, your freeCodeCamp forum leaderboard project is now complete!