Exercise 3 - Async/Await with Promise.all Practice using JSONPlaceholder

Objective:

Your task is to fetch data concurrently from two different endpoints of the JSONPlaceholder API: the users and the posts. You will need to display this data in a structured format on the HTML page. Use the `async/await` syntax along with `Promise.all` to handle these promises concurrently.

You can use some of the code from the fourth exercise in the 'Promises' section.

Requirements:

Part 1: Defining the URLs and Fetch Function

1. **Define the URLs**: Create two constants, `usersURL` and `postsURL`, to store the endpoints for the users and posts from JSONPlaceholder: 'https://jsonplaceholder.typicode.com/users', 'https://jsonplaceholder.typicode.com/posts'.
2. **Define the Fetch Function**: Create an asynchronous function, `fetchData`, which takes a URL and returns the parsed JSON response. Use `await` with the `fetch` method to get the data, and return the parsed JSON.

Part 2: Displaying the Data

3. **Create an Async Function**: Define an async function `displayData` that will be responsible for fetching and displaying the data.

4. **Select the users and posts divs**: Select the users container div from the HTML (with id 'users'). Select the posts container div from the HTML (with id 'posts').

5. **Fetch Users and Posts**: Inside `displayData`, use `Promise.all` along with `await` to concurrently fetch data from both URLs. Destructure the resolved promises to `users` and `posts` variables.

```
const [users, posts] = await Promise.all([fetchData(usersURL),
fetchData(postsURL)]);
```

1. **Handle the Users Data**: After fetching: a. Iterate through the users, creating div elements for each user with the class 'item'. b. Include the user's name and email inside the div, and append it to the users container.
2. **Handle the Posts Data**: Similarly, for the posts data: a. Iterate through the posts, creating div elements for each post with the class 'item'. b. Include the post's title and body inside the div, and append it to the posts container.

Part 3: Error Handling

7. **Handle Errors**: Use a try-catch block to catch any errors that occur during fetching or processing the data: a. In the catch block, select the error container div from the HTML (with id 'error'). b. Set its text content to display an appropriate error message: `An error occurred: ${error.message}. Please try again later.`.

Part 4: Call the Function

8. **Call the `displayData` Function**: Finally, call the `displayData` function to execute the code.

Ways an Error Can Occur:

- Network failure or an incorrect URL.
- The response was not OK (e.g., a 404 Not Found status).
- An error in the processing of the promises.

Tips:

- Embrace the simplicity and readability of `async/await` syntax.
- Remember that `Promise.all` allows you to run multiple promises concurrently.
- Make extensive use of DOM manipulation methods like `getElementById`, `createElement`, `appendChild`, etc.

Expected Outcome:

Upon successful completion, your JavaScript code will fetch users and posts from the JSONPlaceholder API and display them neatly on the webpage using async/await syntax. Any occurring error will be displayed in a designated error section.

Challenge:

Consider adding a loading indicator that displays while the data is being fetched.

This exercise will help you further understand the usage of async/await in conjunction with Promise.all and solidify your understanding of modern asynchronous programming in JavaScript.