JavaScript Challenge 1 - GitHub User Search

Objective:

In this exercise, you will be developing the JavaScript functionality to search and display GitHub user profiles. The HTML and CSS are already provided, and your task is to write the JavaScript code that interacts with the GitHub API to fetch user data and update the user interface accordingly.

Requirements:

Step 1: Define Constant for GitHub API URL

At the start, we need the GitHub API endpoint to fetch user information.

```
const GITHUB_API_URL = 'https://api.github.com/users/';
```

- `GITHUB_API_URL` : This constant holds the URL for the GitHub API endpoint to retrieve user data based on the username.

Step 2: Grab References to HTML Elements

We must target specific HTML elements to interact with the DOM.

```
const userDetailContainer = document.getElementById('user-details');
const searchButton = document.getElementById('searchButton');
```

- `userDetailContainer` : Points to the HTML element with the id 'user-details'. This reference will be used to insert user details into the DOM.
- `searchButton` : Targets the HTML element with the id 'searchButton'. This button will trigger the search functionality.

Step 3: Declare an Asynchronous Function `fetchUser`

Declare an asynchronous function named `fetchUser` , which takes the `username` parameter.

Send a Fetch Request

Inside the function, use the `fetch` method to request the user's details from the provided API URL and username. Handle errors with a try-catch block.

Parse and Display the User Data

Once the response is retrieved, parse it to JSON and pass it to the `displayUser` function.

```
async function fetchUser(username) {
  // code here...
  const user = await response.json();
  displayUser(user);
}
```

Step 4: Create a `displayUser` Function

Write the `displayUser` function to take a `user` parameter and construct the user's HTML.

Constructing User Profile HTML

Define a string and name it `userHTML`. It will contain a template literal of an HTML structure to display the user details (you will find the desired HTML template in the `user.html` file in this exercise folder). Inject to the template literal the user data (e.g., image, name, location, bio, username, website link, and other details) into the HTML.

Update the `user-details` container's `innerHTML` property with the created elements.

```
function displayUser(user) {
  const userHTML = `
    <div class="card grid-2">
      <!-- User details here -->
    </div>
  `;
  userDetailContainer.innerHTML = userHTML;
}
```

Step 5: Add Event Listener for Search Button

Attach a click event listener to the `searchButton`. When clicked, it grabs the entered username from an input field and calls `fetchUser`.

Step 6: Fetch Initial User

As a final step, initially call the `fetchUser` function with a default username (e.g., your GitHub username) to display a default profile when the page loads.

Step 7: User Feedback

Provide feedback to the user if the username doesn't exist or if there is an error in fetching data. You might choose to display an alert, a specific message in the user details container, or another user-friendly notification.

Tips and Considerations:

- Use ES6 syntax and features wherever possible.
- Pay attention to the data in the response JSON, the key names and their values.
- Always think about the user experience, providing clear messages and smooth transitions.
- Use console logs to debug and understand the flow of your code.

Bonus Challenge:

- **Enhanced Error Handling**:
- - If the user performs a search with an empty input, display an error message either as an alert or as a customized error message inside the UI.
  - Instead of simply logging an error to the console, display a friendly error message in the UI, perhaps inside the `userDetailContainer`. This will help in informing the user if something goes wrong. You can consider creating a function to handle this.
- **Loading Indicator**:
  - Implement a loading indicator that shows up while the API request is in progress and hides once the response is received. This gives users feedback that the system is working on their request.
- **Conditional Rendering of User Information:**
  - Modify the function responsible for displaying user data to conditionally render information such as location, bio, website link, etc.
  - Check if the retrieved data for these fields is null or undefined, and if so, do not render the corresponding HTML elements. This helps in displaying only the available information, keeping the UI clean and uncluttered.

- **Search Functionality with Enter Key:**
    - Enhance the search functionality by allowing users to initiate a search by hitting the Enter key within the username input field.
    - Attach an event listener to the input field that listens for the 'keyup' event, and check if the pressed key is the Enter key (event code 'Enter').
    - If the Enter key is detected, call the functions responsible for fetching and displaying the user data, just as you would when clicking the search button.