Exercise 2 - Promise Chaining with Multiple API Calls to JSONPlaceholder

Objective: Fetch data about users and their posts from JSONPlaceholder and display them in the provided HTML structure.

Preparation:

- Review the HTML and CSS files.
- Understand the JSONPlaceholder endpoint, focusing on `/users` and `/posts?userId=USER_ID`.

Steps:

1. **Understand the HTML Structure:**

- Identify the `#users` container, which will hold all user information.
- Analyze how the user and post elements are structured, including classes and IDs.

2. **Set Up the Base URL:**

- Define the base URL constant that you'll append to each specific endpoint.

3. **Fetch Users:**

- **Initiate Fetch Call**: Start by initiating a fetch call to the `/users` endpoint to retrieve the users' data.
- **Handle Response**: Chain a `.then` method to receive the response. Check if the response is OK, otherwise throw an error.
- **Convert Response to JSON**: Call the `response.json()` method to parse the response body into a JSON object.
- **Process Users' Data**: In the next `.then` block, you'll start processing the users' data:
  - **Locate Users' Container**: Using `getElementById` or similar method, locate the container where the user information should be displayed.
  - **Iterate Through Users**: Using `forEach`, iterate through the JSON array of users:
    - **Create User Div**: For each user, create a new `div` element named `userDiv`.

        - **Apply Class**: Add the class name `user` to this element to apply necessary styling.
        - **Structure User Content**: Use a template literal to structure the user's name, email, and an empty container for posts with class `posts`.

```
 userDiv.innerHTML = `
   <h2>${user.name}</h2>
   <p><strong>Email:</strong> ${user.email}</p>
   <div class="posts"></div>
`;
```

        - **Append User Element**: Append this user element to the users' container in the DOM.
    - **Complete User Processing**: At this point, each user's basic information should be appended to the page.

4. Fetch Posts for Each User:

- **Start Fetching Posts Inside User Iteration**: Within the iteration for each user, start another fetch call to the endpoint `/posts?userId=USER_ID`, replacing `USER_ID` with the actual user ID.
- **Handle Posts Response**: Chain a `.then` method to receive the response and convert it into JSON.
- **Process Posts' Data**: In the next `.then`, you'll start processing the posts' data:
    - **Locate Posts' Container**: Use the `querySelector` method on the current user's element to find the container for posts.
    - **Add Posts Header**: Inside this container, insert a header to title the posts section.
    - **Iterate Through Posts**: Using `forEach`, iterate through the JSON array of posts for the current user:
        - **Create Post Div**: Create a new `div` element named `postDiv` for each post.
        - **Apply Class**: Add the class name `post` to this element.
        - **Structure Post Content**: Use a template literal to structure the post's title and body.

```
postDiv.innerHTML = `<strong>${post.title}</strong>
<br>${post.body}`;
```

        - **Append Post Element**: Append this post element to the posts' container within the user's element.

    - **Handle Errors During Post Fetching**: Chain a `.catch` block after processing the posts to handle any errors that occur specifically during the posts fetching process. Use appropriate error handling strategies to log and display any errors.

5. **Error Handling:**

- Implement a `.catch` block at the end of the users' fetch chain to log any errors to the console.

Guidelines:

- Consider creating helper functions to clean up repetitive code, such as generating HTML structures.
- Maintain a clear understanding of the sequence of Promise chaining.
- Regularly test the code in a browser to ensure correct behavior.

Challenge:

- Implement loading indicators that appear during the fetching process.

Note:

- Utilize browser Developer Tools to debug the code and inspect network requests.
- Focus on the structure and flow of Promise chaining, ensuring the code's readability and maintainability.