Async assignment in JavaScript

Introduction

This project showcases different methods of handling asynchronous operations in JavaScript — including callbacks, promises, and async/await. Each method is implemented in a separate HTML page, and all examples involve fetching data from an external API (`https://dummyjson.com/posts`) and displaying it dynamically.

The goal is to demonstrate how asynchronous code works in JavaScript, how to manage delays, handle responses, and deal with errors effectively.

🔗 GitHub Repository:

https://github.com/kvsahithi/fetch-posts

Callback Example

In the **Callback Example**, I demonstrated the use of a traditional callback function with a simulated delay using `setTimeout`.

What I Did:

In `callbacks.html`:

- Created a basic HTML structure with proper `DOCTYPE`, `head`, and `body` tags.

- Included a title and linked an external **favicon** (`favicon.png`) to enhance the visual identity of the page.

- Linked the **CSS file (`styles.css`)** for consistent styling.
- Added a **button** inside a `.button-container` div to trigger the callback-based function.
- Included a `.container` div that contains a heading and an empty `<div id="output">` where the posts will be displayed.
- Linked the external **JavaScript file (`script.js`)** at the bottom.

💻 **In `script.js`:**

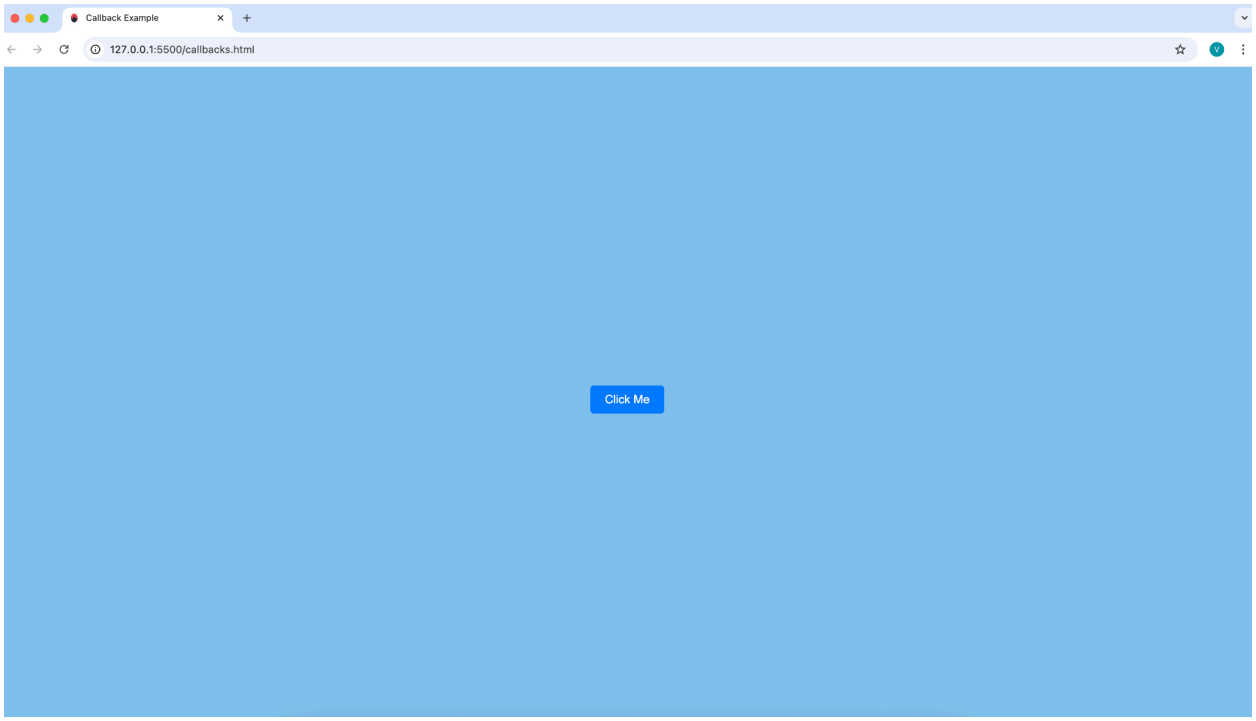
- Defined a function `delayedCallback()` using `setTimeout()` to simulate a 5-second delay before executing a callback function.
- Inside `executeCallback()`, used the delay function and then fetched data from the Dummy JSON Posts API.
- After 5 seconds, the callback triggers a `fetch()` request.
- Retrieved and parsed JSON data and dynamically displayed the **post titles** inside a `<ul>` list in the `#output` div.
- Handled API errors and displayed a friendly error message in case of failure.

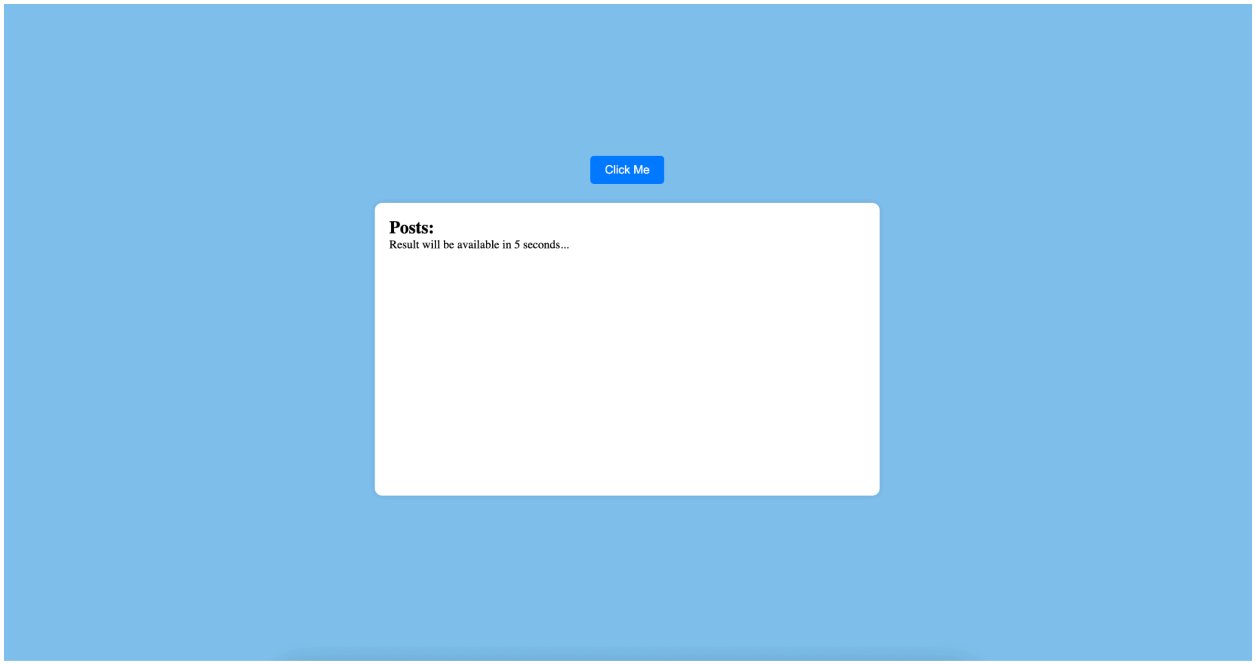
🌐 **What Is Shown on the Webpage:**

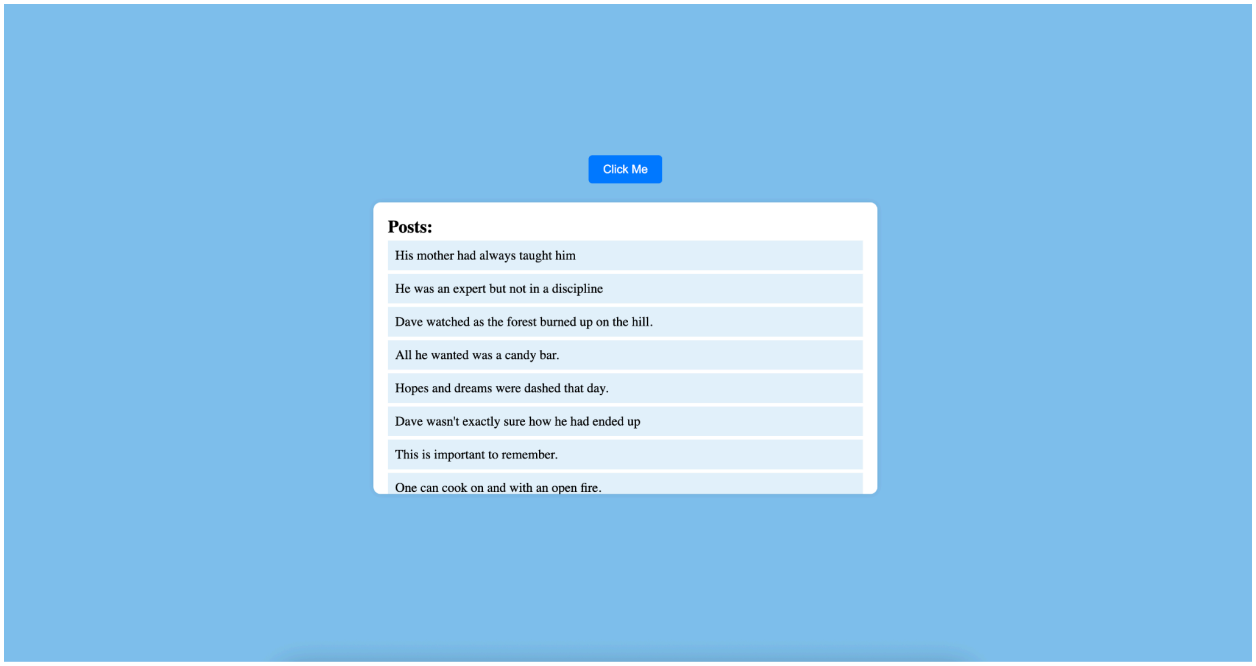
- A button labeled **"Click Me"**.
- When clicked, the user sees a message: **"Result will be available in 5 seconds..."**

- After the delay, the **titles of all posts from the API** are displayed in a styled list.

- If there's a network issue, a red error message appears.







Promises Example

In the **Promises Example**, I demonstrated how to handle asynchronous operations using **JavaScript Promises**, including handling of loading states, successful API responses, and timeout errors.

What I Did:

In `promises.html`:

- Created a basic HTML file with the basic structure of an HTML document.
- Included a page title and linked the **favicon** to maintain visual consistency across all pages.
- Connected the shared **CSS file (`styles.css`)** for layout and design.
- Added a **button** inside a `.button-container` to trigger the Promise-based function.

- Included a `.container` with a heading and a `<div id="result">` to show the fetched posts.

- Linked the **external JavaScript file (`script.js`)** at the bottom of the body.

In `script.js`:

- Defined the function `fetchDataWithPromise()`, which starts by showing **"Loading..."** in the result div.

- Created a custom `Promise` to:

    - Fetch data from the API: https://dummyjson.com/posts

    - Simulate a timeout of **5 seconds** using `setTimeout`. If the request takes too long, the promise is rejected with **"Operation timed out."**

- On successful resolution:

    - Extracted the post titles from the JSON response.

    - Displayed them dynamically inside a `<ul>` list in the `#result` div.

- On error or timeout:

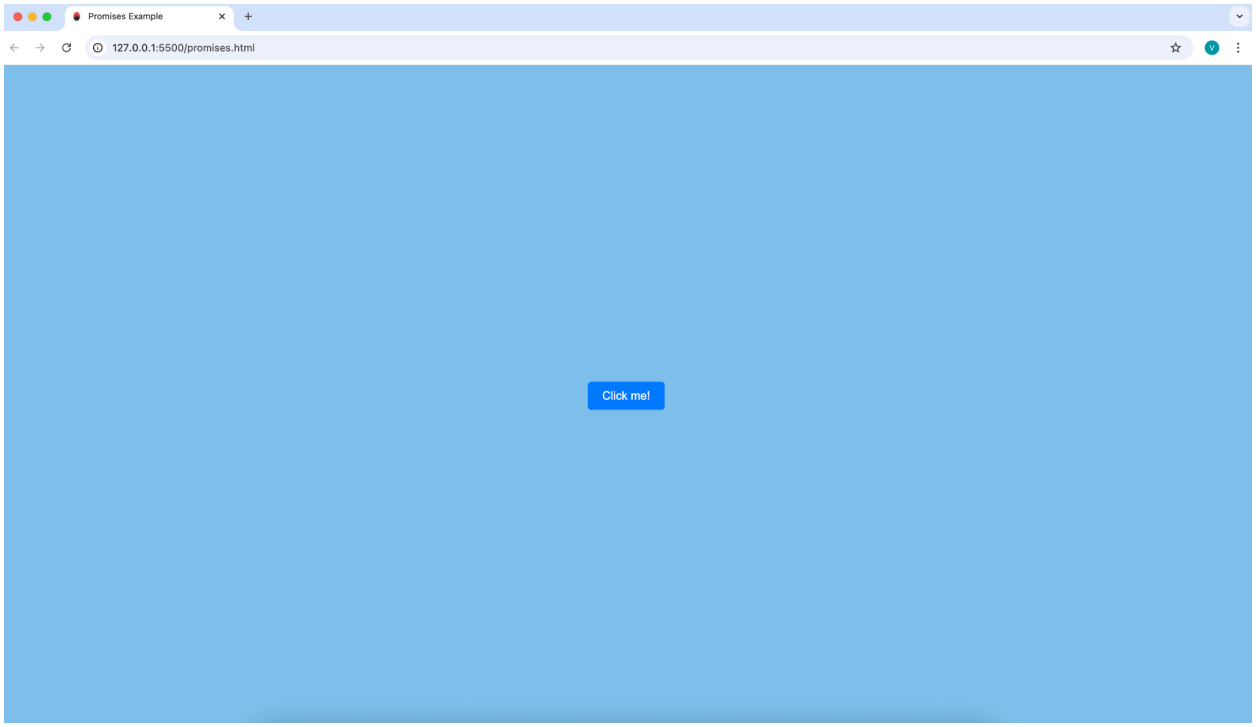
    - Displayed an appropriate error message in red.

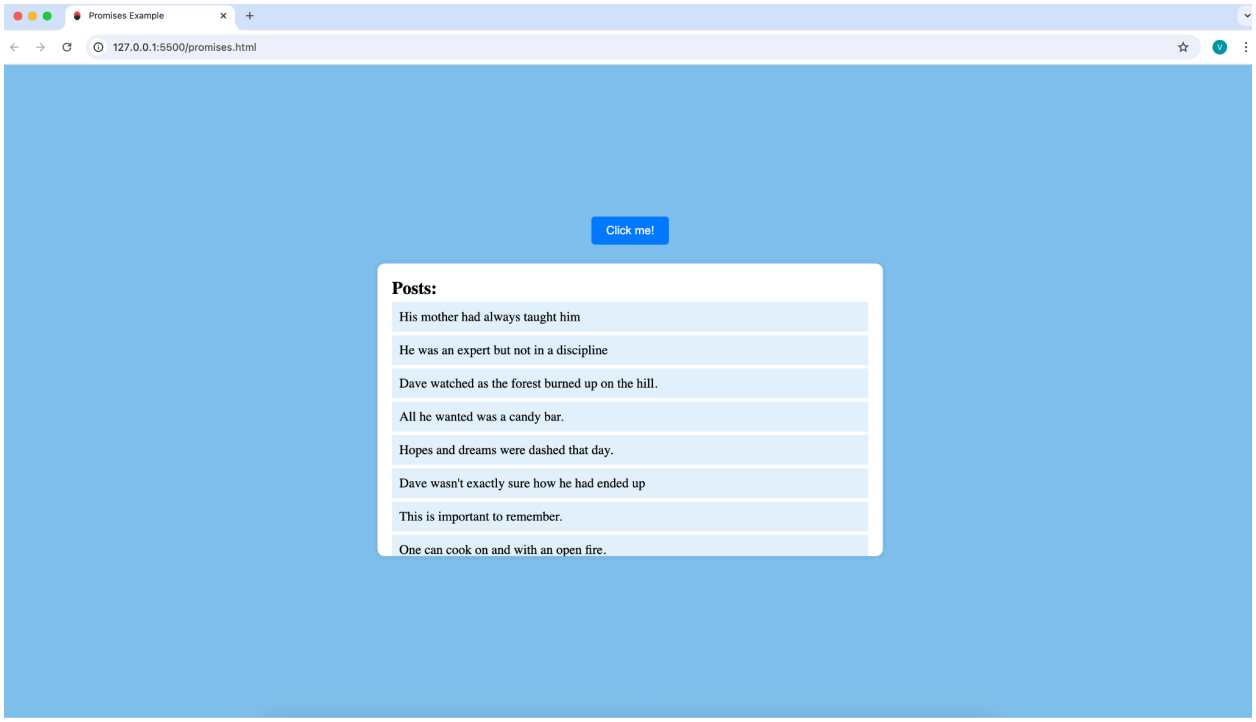
What Is Shown on the Webpage:

- A button labeled **"Click me!"**

- When clicked, the text **"Loading..."** appears while the request is pending.

- If successful, the post titles are shown in a scrollable and styled list.
- If the request fails or takes longer than 5 seconds, a red error message such as **"Operation timed out."** is shown.





Async/Await Example

This example demonstrates the use of the **`async/await`** syntax to handle asynchronous operations in a more readable and cleaner way compared to callbacks or traditional Promises.

What I Did:

In `async-await.html`:

- Created a well-structured HTML document with:
    - A descriptive `<title>` tag.
    - Linked a **favicon** to visually brand the page.
    - Linked an external **CSS file (`styles.css`)** for consistent design.

- Included a **button** labeled **"Click me!"** which calls the `fetchDataWithAsync()` function when clicked.

- Added a `.container` section to display the fetched post data.

- Linked the **JavaScript file (`script.js`)** just before the closing `</body>` tag.

In `script.js`:

- Defined an `async` function `fetchDataWithAsync()` to fetch post data from the API:
  https://dummyjson.com/posts

- Displayed a **"Loading..."** message while waiting for the response.

- Used **`AbortController`** to automatically cancel the request if it takes longer than **5 seconds**, satisfying timeout handling.

- On success, the function:

    - Parses the returned JSON.

    - Calls `displayPosts()` to render post titles in a clean list.

- On failure (including timeout), it:

    - Catches the error.

    - Displays an appropriate error message inside the output div.

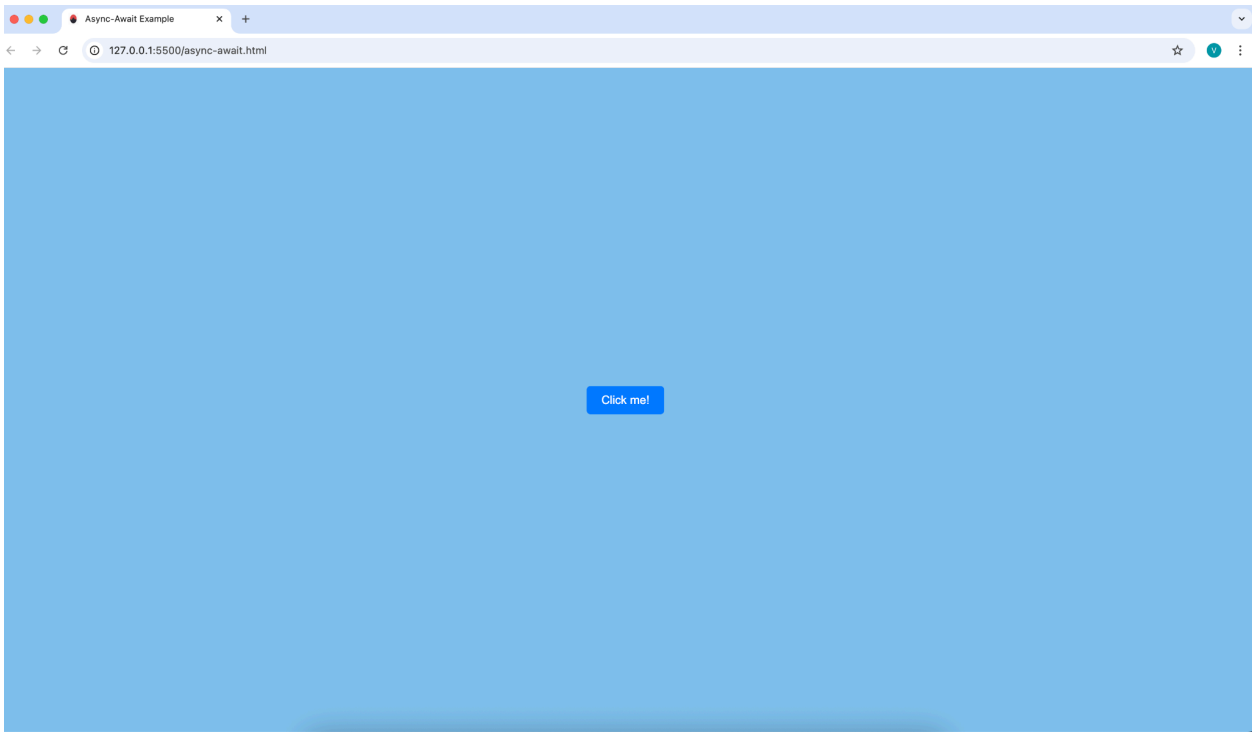
- `displayPosts()` is a helper function that:

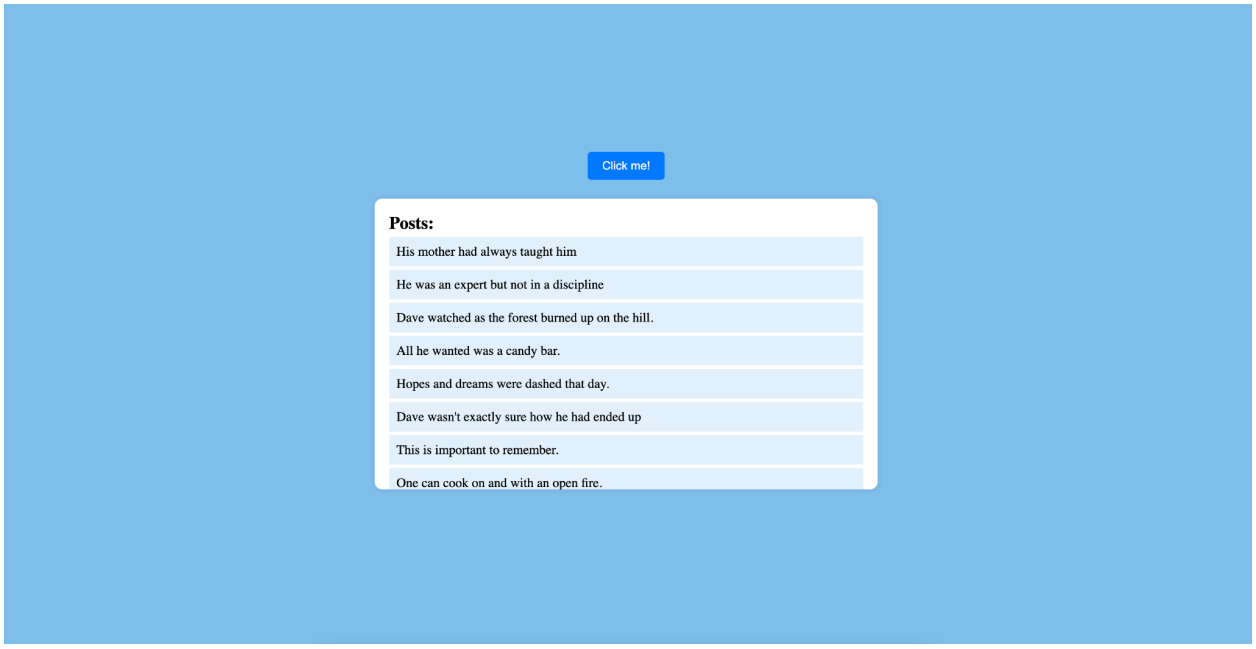
    - Loops through the `posts` array.

- Creates a styled list of post titles and injects it into the page.

🌐 What Is Shown on the Webpage:

- A **button** that initiates the data fetching process.
- When clicked:
  - Shows **"Loading..."** immediately.
  - After the data is successfully fetched, it displays a list of **post titles** from the API.
  - If an error or timeout occurs, a clear error message (in red) is shown.





This completes the assignment. All tasks were done as per the given instructions using callbacks, promises, and async/await.