Project Documentation: Asynchronous JavaScript Demonstration

Project Overview

This project demonstrates three approaches to handling asynchronous operations in JavaScript:

1. **Callbacks**
2. **Promises**
3. **Async/Await**

Each method fetches posts from an API and dynamically displays them on the webpage.

Technologies Used

- **Frontend:** HTML, CSS, JavaScript
- **API:** DummyJSON for fetching posts
- **Styles:** Custom CSS for UI elements

Project Structure

```
/project-folder
│── index.html        // Main entry page
│── index.css         // Styling for all pages
│── callbacks.html    // Page demonstrating Callbacks
│── promises.html     // Page demonstrating Promises
│── async-await.html  // Page demonstrating Async/Await
│── callbacks.js      // Callback-based asynchronous implementation
│── promises.js       // Promise-based asynchronous implementation
│── async-await.js    // Async/Await-based asynchronous implementation
```

Explanation of Key Files

HTML Files

`index.html` - Home Page

- Provides navigation to explore Callbacks, Promises, and Async/Await pages.
- Contains a welcoming message with brief explanations.

`callbacks.html` - Demonstrating Callbacks

- Includes a button that, when clicked, fetches and displays posts using a callback-based approach.
- Uses JavaScript to handle asynchronous operations via callbacks.

`promises.html` - Demonstrating Promises

- Includes a button to fetch and display posts using Promises.
- Uses `.then()` and `.catch()` to manage API calls.

`async-await.html` - Demonstrating Async/Await

- Includes a button to fetch and display posts using Async/Await.
- Uses modern JavaScript async functions for better readability.

JavaScript Files

`callbacks.js`

- Fetches data using the traditional callback pattern.
- Implements a timeout to simulate delayed execution before fetching posts.
- Provides error handling for failed requests.
- Dynamically renders posts inside an unordered list (`<ul>`).

`promises.js`

- Uses Promises to fetch and display posts.
- Implements `.then()` and `.catch()` for handling responses and errors.
- Includes a timeout mechanism to reject the Promise if the request takes too long.
- Dynamically creates and displays posts in a structured format.

`async-await.js`

- Uses `async/await` syntax for fetching posts.
- Implements an `AbortController` to handle timeouts for API requests.
- Provides better error handling and readable code structure.
- Dynamically updates the UI with posts retrieved from the API.

CSS File

`index.css`

- Defines styles for the UI elements, including buttons, navigation, and post display.
- Implements responsive design for better user experience.
- Styles buttons with hover effects and smooth transitions.
- Designs dynamic post cards with toggleable content for expanding/collapsing text.
- Includes error message styling for failed API requests.

Features

- Interactive UI with buttons to trigger API calls.
- Dynamic rendering of posts fetched from the API.
- Error handling for API failures.
- Three different approaches to handling asynchronous operations.
- Loader animations to indicate data fetching.
- Toggle functionality for viewing full post content.

How to Use the Project

1. Open `index.html` in a browser.
2. Click on any of the navigation links (`Callbacks`, `Promises`, or `Async/Await`).
3. Click the button on the respective page to fetch and display posts.
4. Observe how each approach handles asynchronous operations differently.
5. Click on "View More" to expand post content and "View Less" to collapse it.

Project Links

- **Live Project:** https://async-javascript.netlify.app/
- **GitHub Repository:** https://github.com/SravanGunaganti/Async-Javascript.git

Conclusion

This project effectively demonstrates the differences between **Callbacks, Promises, and Async/Await** in JavaScript. By understanding these approaches, developers can choose the best method based on readability, efficiency, and error handling.