Blog exercise with 2 connected API requests

In this exercise, you will build a simple blog application that fetches data from an API and displays it to users. The application should allow users to view a list of blog posts and click on individual posts to read the full content.

Requirements

Your blog application must include the following features:

- **Display all blog posts** - Fetch and show a list of all available blog posts
- **Show post preview** - Display the title, a text excerpt, and date for each post
- **Clickable posts** - Make each post clickable to view the full content
- **Individual post page** - Display the complete post information including:
  - Full title
  - Complete text content
  - Publication date
  - Image (if available)
  - Tags (if available)
- **Navigation** - Provide a way to navigate back to the blog list from individual posts
- **Error handling** - Display appropriate error messages if data fetching fails

API Endpoints

Use the following API endpoints to fetch blog data:

1. Get All Blog Posts

URL: `https://codexplained.se/api/blog_json.php`

Returns: An array containing a list of all blog posts. Each post in the list has an ID that you'll need to fetch the specific post details.

Usage: Fetch this list and display all posts. Each post should be clickable to navigate to the individual post page.

2. Get Individual Blog Post

URL: `https://codexplained.se/api/post_json.php?post={postId}`

Parameter: `post` - The ID of the post to retrieve

Example: `https://codexplained.se/api/post_json.php?post=1`

Returns: Complete information about a single blog post

Usage: When a user clicks on a post from the list, use the post's ID to fetch and display its full details using this endpoint.

Implementation Approach

You can choose one of the following approaches:

Option 1: Single Page Application (One HTML file)

Create a single HTML page that dynamically switches between showing the list of posts and showing individual post details.

Option 2: Multi-Page Application (Separate files)

Create two separate HTML pages:

- `index.html` - Displays the list of all blog posts
- `post.html` - Displays a single post (uses URL parameters to get post ID)

Suggested Steps

> **Step 1: Create the HTML structure**

Set up your HTML file(s) with containers for displaying posts. Include basic CSS for styling.

Step 2: Fetch all blog posts

Write a JavaScript function that fetches all posts from the API using `fetch()` and the `async/await` syntax.

Step 3: Display posts list

Create a function that takes the posts array and dynamically generates HTML to display each post as a card or list item.

Step 4: Make posts clickable

Add click handlers or links to each post that will load the individual post details.

Step 5: Fetch and display individual post

Create functionality to fetch a specific post by ID and display its complete content.

Step 6: Add navigation

Include a back button or link to return to the blog list.

Step 7: Add error handling

Use try-catch blocks to handle errors and display user-friendly error messages.

💡 Hints

- Use `fetch()` with `async/await` for cleaner asynchronous code

- Use `window.addEventListener('DOMContentLoaded', ...)` to run code when the page loads
- For multi-page approach, use `URLSearchParams` to get the post ID from the URL query string
- For single-page approach, use CSS classes like `.hidden` to show/hide sections
- Remember to handle cases where data might be missing (e.g., no image, no tags)