**Fetch and Display Data**

1. To implement a JavaScript function that fetches and displays data, we'll use the fetch() API to retrieve data from a remote server or an API and then display it dynamically on a webpage. For simplicity, let's assume we're fetching data from a public API.

**Example: Fetch and Display Data from a Public API**

In this example, we'll fetch data from the JSONPlaceholder API, which is a free online REST API for testing and prototyping.

**Code Example:**

html

CopyEdit

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Fetch and Display Data</title>

<style>

body {

font-family: Arial, sans-serif;

text-align: center;

margin-top: 50px;

}

.data-container {

margin-top: 20px;

}

.data-item {

margin: 10px;

padding: 10px;

background-color: #f0f0f0;

border-radius: 5px;

}

.loading {

font-size: 18px;

color: #555;

}

.error {

color: red;

font-size: 18px;

}

</style>

</head>

<body>

<h1>Fetch and Display Data</h1>

<button onclick="fetchData()">Fetch Data</button>

<div id="message" class="loading">Click the button to load data...</div>

<div id="data-container" class="data-container"></div>

<script>

// Function to fetch and display data

async function fetchData() {

const messageElement = document.getElementById('message');

const dataContainer = document.getElementById('data-container');

messageElement.textContent = 'Loading data...'; // Display loading message

try {

// Fetching data from JSONPlaceholder API

const response = await fetch('https://jsonplaceholder.typicode.com/posts');

if (!response.ok) {

throw new Error('Failed to fetch data');

}

// Parse JSON data

const data = await response.json();

// Clear any previous content

dataContainer.innerHTML = '';

// Loop through data and display each item

data.slice(0, 5).forEach(item => { // Limiting to first 5 posts

const div = document.createElement('div');

div.className = 'data-item';

div.innerHTML = `<h3>${item.title}</h3><p>${item.body}</p>`;

dataContainer.appendChild(div);

});

// Update the message after data is loaded

messageElement.textContent = 'Data loaded successfully!';

} catch (error) {

// Handle errors

messageElement.textContent = 'Error loading data.';

messageElement.classList.add('error');

console.error('Error fetching data:', error);

}

}

</script>

</body>

</html>

**Explanation:**

1. **HTML**: We have a button that triggers the fetchData() function when clicked. There's also a div with id="message" to display loading or error messages and a div with id="data-container" where the fetched data will be displayed.
2. **CSS**: Provides some basic styling to the button, data items, and loading/error messages.
3. **JavaScript**:
   * The fetchData() function is an async function, meaning it can handle asynchronous code with await.
   * It starts by setting the message to "Loading data..." and then makes an HTTP GET request using fetch() to retrieve data from the JSONPlaceholder API (https://jsonplaceholder.typicode.com/posts).
   * Once the data is fetched, it's parsed into a JSON format using .json().
   * We then loop through the fetched data, creating div elements to display each post's title and body. These are appended to the data-container element.
   * If there's an error during the fetch, an error message is displayed in red.
4. **Error Handling**: If the fetch request fails (e.g., network issues), an error message will be shown. This is handled by the try-catch block.

**How it Works:**

* When the user clicks the "Fetch Data" button, the fetchData() function is called.
* The page displays "Loading data..." while the data is being fetched.
* Once the data is fetched and parsed, it displays the first 5 posts from the API, showing each post's title and body.
* If there's an error (e.g., network failure), an error message is displayed.

This is a simple example, but it demonstrates how to interact with an API, handle asynchronous code, and update the webpage dynamically based on the fetched data.