Chapter 11

API Integration and Fetching Data

**Example of a RESTful API Request**

Let’s look at an example of a simple API that provides information about users.

* **URL**: https://jsonplaceholder.typicode.com/users
* **HTTP Method**: GET
* **Response** (JSON):

[

{

"id": 1,

"name": "Leanne Graham",

"username": "Bret",

"email": "Sincere@april.biz",

"address": {

"street": "Kulas Light",

"suite": "Apt. 556",

"city": "Gwenborough",

"zipcode": "92998-3874",

"geo": {

"lat": "-37.3159",

"lng": "81.1496"

}

},

"phone": "1-770-736-8031 x56442",

"website": "hildegard.org",

"company": {

"name": "Romaguera-Crona",

"catchPhrase": "Multi-layered client-server neural-net",

"bs": "harness real-time e-markets"

}

},

{

"id": 2,

"name": "Ervin Howell",

"username": "Antonette",

"email": "Shanna@melissa.tv",

"address": {

"street": "Victor Plains",

"suite": "Suite 879",

"city": "Wisokyburgh",

"zipcode": "90566-7771",

"geo": {

"lat": "-43.9509",

"lng": "-34.4618"

}

},

"phone": "010-692-6593 x09125",

"website": "anastasia.net",

"company": {

"name": "Deckow-Crist",

"catchPhrase": "Proactive didactic contingency",

"bs": "synergize scalable supply-chains"

}

},

{

"id": 3,

"name": "Clementine Bauch",

"username": "Samantha",

"email": "Nathan@yesenia.net",

"address": {

"street": "Douglas Extension",

"suite": "Suite 847",

"city": "McKenziehaven",

"zipcode": "59590-4157",

"geo": {

"lat": "-68.6102",

"lng": "-47.0653"

}

},

"phone": "1-463-123-4447",

"website": "ramiro.info",

"company": {

"name": "Romaguera-Jacobson",

"catchPhrase": "Face to face bifurcated interface",

"bs": "e-enable strategic applications"

}

},

{

"id": 4,

"name": "Patricia Lebsack",

"username": "Karianne",

"email": "Julianne.OConner@kory.org",

"address": {

"street": "Hoeger Mall",

"suite": "Apt. 692",

"city": "South Elvis",

"zipcode": "53919-4257",

"geo": {

"lat": "29.4572",

"lng": "-164.2990"

}

},

"phone": "493-170-9623 x156",

"website": "kale.biz",

"company": {

"name": "Robel-Corkery",

"catchPhrase": "Multi-tiered zero tolerance productivity",

"bs": "transition cutting-edge web services"

}

},

{

"id": 5,

"name": "Chelsey Dietrich",

"username": "Kamren",

"email": "Lucio\_Hettinger@annie.ca",

"address": {

"street": "Skiles Walks",

"suite": "Suite 351",

"city": "Roscoeview",

"zipcode": "33263",

"geo": {

"lat": "-31.8129",

"lng": "62.5342"

}

},

"phone": "(254)954-1289",

"website": "demarco.info",

"company": {

"name": "Keebler LLC",

"catchPhrase": "User-centric fault-tolerant solution",

"bs": "revolutionize end-to-end systems"

}

},

{

"id": 6,

"name": "Mrs. Dennis Schulist",

"username": "Leopoldo\_Corkery",

"email": "Karley\_Dach@jasper.info",

"address": {

"street": "Norberto Crossing",

"suite": "Apt. 950",

"city": "South Christy",

"zipcode": "23505-1337",

"geo": {

"lat": "-71.4197",

"lng": "71.7478"

}

},

"phone": "1-477-935-8478 x6430",

"website": "ola.org",

"company": {

"name": "Considine-Lockman",

"catchPhrase": "Synchronised bottom-line interface",

"bs": "e-enable innovative applications"

}

},

{

"id": 7,

"name": "Kurtis Weissnat",

"username": "Elwyn.Skiles",

"email": "Telly.Hoeger@billy.biz",

"address": {

"street": "Rex Trail",

"suite": "Suite 280",

"city": "Howemouth",

"zipcode": "58804-1099",

"geo": {

"lat": "24.8918",

"lng": "21.8984"

}

},

"phone": "210.067.6132",

"website": "elvis.io",

"company": {

"name": "Johns Group",

"catchPhrase": "Configurable multimedia task-force",

"bs": "generate enterprise e-tailers"

}

},

{

"id": 8,

"name": "Nicholas Runolfsdottir V",

"username": "Maxime\_Nienow",

"email": "Sherwood@rosamond.me",

"address": {

"street": "Ellsworth Summit",

"suite": "Suite 729",

"city": "Aliyaview",

"zipcode": "45169",

"geo": {

"lat": "-14.3990",

"lng": "-120.7677"

}

},

"phone": "586.493.6943 x140",

"website": "jacynthe.com",

"company": {

"name": "Abernathy Group",

"catchPhrase": "Implemented secondary concept",

"bs": "e-enable extensible e-tailers"

}

},

{

"id": 9,

"name": "Glenna Reichert",

"username": "Delphine",

"email": "Chaim\_McDermott@dana.io",

"address": {

"street": "Dayna Park",

"suite": "Suite 449",

"city": "Bartholomebury",

"zipcode": "76495-3109",

"geo": {

"lat": "24.6463",

"lng": "-168.8889"

}

},

"phone": "(775)976-6794 x41206",

"website": "conrad.com",

"company": {

"name": "Yost and Sons",

"catchPhrase": "Switchable contextually-based project",

"bs": "aggregate real-time technologies"

}

},

{

"id": 10,

"name": "Clementina DuBuque",

"username": "Moriah.Stanton",

"email": "Rey.Padberg@karina.biz",

"address": {

"street": "Kattie Turnpike",

"suite": "Suite 198",

"city": "Lebsackbury",

"zipcode": "31428-2261",

"geo": {

"lat": "-38.2386",

"lng": "57.2232"

}

},

"phone": "024-648-3804",

"website": "ambrose.net",

"company": {

"name": "Hoeger LLC",

"catchPhrase": "Centralized empowering task-force",

"bs": "target end-to-end models"

}

}

]

**Syntax of fetch()**

The basic syntax of fetch() looks like this:

fetch(url, options)

.then(response => response.json())

.then(data => {

// Process the data

})

.catch(error => {

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

});

**Basic Example of a GET Request**

Here's an example of making a simple GET request to fetch data from an API.

fetch('https://jsonplaceholder.typicode.com/posts/1')

.then(response => response.json()) // Convert the response to JSON

.then(data => {

console.log(data); // Log the fetched data to the console

})

.catch(error => {

console.error('Error:', error); // Handle errors

});

**Complete Code:**

**HTML:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Fetch API Example</title>

<style>

body {

font-family: Arial, sans-serif;

margin: 20px;

}

.user-card {

border: 1px solid #ccc;

padding: 10px;

margin: 10px 0;

border-radius: 5px;

box-shadow: 2px 2px 8px rgba(0, 0, 0, 0.1);

}

.user-card h3 {

margin: 0 0 5px;

}

.user-card p {

margin: 0;

}

</style>

</head>

<body>

<h1>Users List</h1>

<div id="user-list"></div>

<script>

// Define the API endpoint and options

const url = 'https://jsonplaceholder.typicode.com/users';

const options = {

method: 'GET', // Default method

headers: {

'Content-Type': 'application/json'

}

};

// Fetch data from the API

fetch(url, options)

.then(response => {

if (!response.ok) {

throw new Error(`HTTP error! status: ${response.status}`);

}

return response.json();

})

.then(data => {

// Process and display the data

const userList = document.getElementById('user-list');

data.forEach(user => {

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

userCard.className = 'user-card';

userCard.innerHTML = `

<h3>${user.name}</h3>

<p><strong>Email:</strong> ${user.email}</p>

<p><strong>Phone:</strong> ${user.phone}</p>

<p><strong>Company:</strong> ${user.company.name}</p>

`;

userList.appendChild(userCard);

});

})

.catch(error => {

// Handle errors

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

const userList = document.getElementById('user-list');

userList.innerHTML = `<p style="color: red;">Failed to load data. Please try again later.</p>`;

});

</script>

</body>

</html>

**Making a POST Request**

To send data to the server (e.g., creating a new post), you use the POST method with fetch(). Here's an example:

const postData = {

title: 'New Post',

body: 'This is a new post.',

userId: 1

};

fetch('https://jsonplaceholder.typicode.com/posts', {

method: 'POST', // Specify the HTTP method

headers: {

'Content-Type': 'application/json' // Specify the content type of the request body

},

body: JSON.stringify(postData) // Convert the JavaScript object to JSON format

})

.then(response => response.json()) // Convert the response to JSON

.then(data => {

console.log('Post created:', data); // Log the response data

})

.catch(error => {

console.error('Error:', error); // Handle errors

});

**Working with Query Parameters in GET Requests**

Often, when making a GET request, you need to pass query parameters in the URL. Here's an example of how to handle query parameters with fetch():

const userId = 1;

fetch(`https://jsonplaceholder.typicode.com/posts?userId=${userId}`)

.then(response => response.json())

.then(data => {

console.log(data); // Log the data filtered by userId

})

.catch(error => {

console.error('Error:', error); // Handle errors

});

**Handling Different Response Status Codes**

Sometimes, the server might return a non-200 status code, indicating an error (e.g., 404, 500). To handle such cases, you can check the response.ok property, which is true for successful responses and false for errors.

fetch('https://jsonplaceholder.typicode.com/posts/9999') // Non-existent post

.then(response => {

if (!response.ok) {

throw new Error('Post not found');

}

return response.json(); // Convert to JSON if successful

})

.then(data => {

console.log(data); // Log the fetched data

})

.catch(error => {

console.error('Error:', error.message); // Handle errors

});

* In this example:
  + The response.ok property is checked to determine if the request was successful.
  + If the status code is not 200, an error is thrown with a custom message.

**Handling Headers**

You can also include headers in your request to provide additional information, such as authentication tokens or content type.

fetch('https://jsonplaceholder.typicode.com/posts', {

method: 'GET',

headers: {

'Authorization': 'Bearer your-token-here',

'Accept': 'application/json'

}

})

.then(response => response.json())

.then(data => {

console.log(data); // Log the data fetched

})

.catch(error => {

console.error('Error:', error); // Handle errors

});

* In this example:
  + The request includes an Authorization header with a Bearer token.
  + The Accept header specifies that the client expects a response in JSON format.

**How JSON is Returned in API Responses**

APIs often return data in JSON format in response to requests. This data can represent various types of information, such as user data, posts, comments, etc. Here's an example of what a typical JSON response might look like:

{

"id": 1,

"title": "Hello World",

"body": "This is a simple API response example.",

"userId": 1

}

This JSON object contains properties like id, title, body, and userId.

**Example of Handling API Responses and Parsing JSON**

Here’s an example where we make a GET request to fetch data from an API and parse the JSON response:

fetch('https://jsonplaceholder.typicode.com/posts/1') // URL of the API endpoint

.then(response => {

if (!response.ok) { // Check if response status is OK (200-299)

throw new Error('Network response was not ok ' + response.statusText);

}

return response.json(); // Parse the response to JSON

})

.then(data => {

console.log('Fetched Data:', data); // Log the parsed JSON object

})

.catch(error => {

console.error('There was a problem with the fetch operation:', error); // Handle errors

});

**Example of Handling Nested JSON Objects**

Sometimes, API responses may include nested objects. For example, you might get a response like this:

{

"post": {

"id": 1,

"title": "Hello World",

"body": "This is a simple API response example.",

"user": {

"id": 1,

"name": "John Doe",

"email": "john.doe@example.com"

}

}

}

In this case, the post object contains another object user. To access nested data, you can simply use dot notation.

fetch('https://jsonplaceholder.typicode.com/posts/1')

.then(response => response.json())

.then(data => {

console.log('Post Title:', data.post.title); // Access nested data

console.log('User Name:', data.post.user.name); // Access nested user data

})

.catch(error => {

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

});

**Handling Errors in JSON Parsing**

fetch('https://jsonplaceholder.typicode.com/posts/1')

.then(response => {

if (!response.ok) {

throw new Error('Network response was not ok ' + response.statusText);

}

return response.text(); // Get response as plain text

})

.then(text => {

try {

const data = JSON.parse(text); // Manually parse the text to JSON

console.log(data);

} catch (error) {

console.error('Error parsing JSON:', error); // Handle JSON parsing errors

}

})

.catch(error => {

console.error('There was a problem with the fetch operation:', error); // Handle other errors

});

**Example of Network Error Handling**

fetch('https://jsonplaceholder.typicode.com/users') // Alternate real API URL

.then(response => {

if (!response.ok) { // Check if the response status is OK (2xx)

throw new Error('Network response was not ok');

}

return response.json(); // Parse the response to JSON

})

.then(data => {

console.log('Fetched Data:', data); // Process and log the response data

// Optionally, you can manipulate the data here or display it on a webpage

})

.catch(error => {

console.error('There was a problem with the fetch operation:', error); // Handle network errors

});

**Example of Handling Response Errors**

fetch('https://jsonplaceholder.typicode.com/posts/1') // Example API endpoint

.then(response => {

if (!response.ok) { // Check if the status code is outside the range of 200-299

if (response.status === 404) {

throw new Error('Post not found');

} else if (response.status === 500) {

throw new Error('Server error, please try again later');

} else {

throw new Error('Something went wrong');

}

}

return response.json(); // Parse the response as JSON if the request was successful

})

.then(data => {

console.log('Post Data:', data);

})

.catch(error => {

console.error('Error:', error.message); // Log the error message for debugging or display it to the user

});

**Example of Handling JSON Parsing Errors**

fetch('https://jsonplaceholder.typicode.com/posts/1')

.then(response => response.text()) // Get the response as plain text

.then(text => {

try {

const data = JSON.parse(text); // Attempt to parse the response as JSON

console.log('Parsed Data:', data);

} catch (error) {

throw new Error('Failed to parse JSON');

}

})

.catch(error => {

console.error('Error:', error.message);

});

**Example of Error Handling with async/await**

async function fetchData() {

try {

const response = await fetch('https://jsonplaceholder.typicode.com/posts/1');

if (!response.ok) { // Check if the response status is OK

throw new Error('Network response was not ok');

}

const data = await response.json(); // Parse the JSON response

console.log('Post Data:', data);

} catch (error) {

console.error('Error:', error.message); // Handle errors from fetch or JSON parsing

}

}

fetchData();

**Example: Fetching Data from a Weather API**

To fetch weather data, you can use the OpenWeatherMap API. First, sign up to obtain an API key.

const apiKey = 'YOUR\_API\_KEY';

const city = 'London';

const url = `https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${apiKey}`;

fetch(url)

.then(response => {

if (!response.ok) {

throw new Error(`HTTP error! status: ${response.status}`);

}

return response.json();

})

.then(data => {

console.log('Weather Data:', data);

// Process and display the weather data

})

.catch(error => {

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

});

**Example: Fetching Data from the GitHub API**

To fetch user information from GitHub:

const username = 'octocat';

const url = `https://api.github.com/users/${username}`;

fetch(url)

.then(response => {

if (!response.ok) {

throw new Error(`HTTP error! status: ${response.status}`);

}

return response.json();

})

.then(data => {

console.log('GitHub User Data:', data);

// Process and display the user data

})

.catch(error => {

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

});

**Handling API Responses**

API responses are typically in JSON format. After parsing the JSON data, you can manipulate it as needed.

fetch(url)

.then(response => response.json())

.then(data => {

// Access specific data fields

const temperature = data.main.temp;

const weatherDescription = data.weather[0].description;

console.log(`Temperature: ${temperature}`);

console.log(`Weather: ${weatherDescription}`);

});

**Error Handling**

Proper error handling ensures your application can gracefully manage issues during API requests.

fetch(url)

.then(response => {

if (!response.ok) {

throw new Error(`HTTP error! status: ${response.status}`);

}

return response.json();

})

.catch(error => {

console.error('There was a problem with the fetch operation:', error);

// Display a user-friendly message or retry the request

});

**Fetching Data with Promises**

A **Promise** is an object that represents the eventual completion (or failure) of an asynchronous operation. The fetch() function returns a Promise that resolves once the HTTP request is completed.

**Basic Syntax**

fetch(url)

.then(response => {

if (!response.ok) {

throw new Error('Network response was not ok');

}

return response.json(); // Parse JSON data

})

.then(data => {

console.log('Data fetched:', data); // Handle the fetched data

})

.catch(error => {

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

});

**Example: Fetching User Data from an API**

const url = 'https://jsonplaceholder.typicode.com/users';

fetch(url)

.then(response => {

if (!response.ok) {

throw new Error(`HTTP error! Status: ${response.status}`);

}

return response.json();

})

.then(users => {

users.forEach(user => console.log(`Name: ${user.name}`));

})

.catch(error => {

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

});

**Understanding async/await**

**Fetching Data with async/await**

**Basic Syntax**

async function fetchData(url) {

try {

const response = await fetch(url); // Wait for the fetch to resolve

if (!response.ok) {

throw new Error(`HTTP error! Status: ${response.status}`);

}

const data = await response.json(); // Wait for JSON parsing

console.log('Data fetched:', data);

} catch (error) {

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

}

}

**Example: Fetching Posts from an API**

const url = 'https://jsonplaceholder.typicode.com/posts';

async function fetchPosts() {

try {

const response = await fetch(url); // Make the HTTP request

if (!response.ok) {

throw new Error(`HTTP error! Status: ${response.status}`);

}

const posts = await response.json(); // Parse JSON

posts.forEach(post => console.log(`Title: ${post.title}`));

} catch (error) {

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

}

}

fetchPosts();

**Error Handling with async/await**

**Example: Handling API Errors**

async function fetchWithErrors(url) {

try {

const response = await fetch(url);

if (!response.ok) {

throw new Error(`HTTP error! Status: ${response.status}`);

}

const data = await response.json();

console.log('Fetched Data:', data);

} catch (error) {

console.error('Error occurred:', error);

}

}

fetchWithErrors('https://jsonplaceholder.typicode.com/invalidEndpoint');

**Sequential Requests**

async function fetchSequentially() {

try {

const userResponse = await fetch('https://jsonplaceholder.typicode.com/users/1');

const user = await userResponse.json();

const postsResponse = await fetch(`https://jsonplaceholder.typicode.com/posts?userId=${user.id}`);

const posts = await postsResponse.json();

console.log('User:', user);

console.log('Posts:', posts);

} catch (error) {

console.error('Error in sequential fetch:', error);

}

}

**Concurrent Requests with Promise.all**

async function fetchConcurrently() {

try {

const [users, posts] = await Promise.all([

fetch('https://jsonplaceholder.typicode.com/users').then(res => res.json()),

fetch('https://jsonplaceholder.typicode.com/posts').then(res => res.json())

]);

console.log('Users:', users);

console.log('Posts:', posts);

} catch (error) {

console.error('Error in concurrent fetch:', error);

}

}

**Setting Up the HTML Structure**

The webpage should include elements to display the fetched data. For example:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Dynamic Data Rendering</title>

<style>

.user {

margin: 10px 0;

padding: 10px;

border: 1px solid #ccc;

border-radius: 5px;

}

</style>

</head>

<body>

<h1>Dynamic User List</h1>

<div id="user-container"></div>

<script src="script.js"></script>

</body>

</html>

The #user-container div will act as a placeholder for the dynamically rendered user data.

**Fetching Data from an API**

In JavaScript, use the fetch() API or libraries like Axios to fetch data. Here's an example:

const apiUrl = 'https://jsonplaceholder.typicode.com/users';

fetch(apiUrl)

.then(response => {

if (!response.ok) {

throw new Error(`HTTP error! Status: ${response.status}`);

}

return response.json(); // Parse JSON data

})

.then(data => {

renderUsers(data); // Pass data to the rendering function

})

.catch(error => {

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

});

**Rendering Data Dynamically**

Once the data is fetched, use JavaScript to update the DOM. Here’s how you can render a list of users:

function renderUsers(users) {

const container = document.getElementById('user-container');

container.innerHTML = ''; // Clear existing content if any

users.forEach(user => {

// Create a new div for each user

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

userDiv.className = 'user';

userDiv.innerHTML = `

<h3>${user.name}</h3>

<p>Email: ${user.email}</p>

<p>Phone: ${user.phone}</p>

`;

container.appendChild(userDiv); // Append the div to the container

});

}

**Styling the Rendered Content**

CSS can be used to style the dynamically rendered content. For example:

.user {

margin: 10px 0;

padding: 10px;

border: 1px solid #ccc;

border-radius: 5px;

}

.user h3 {

margin: 0 0 5px;

}

.user p {

margin: 0;

}

**Error Handling**

Handle errors gracefully by providing user feedback. For instance:

function renderError(message) {

const container = document.getElementById('user-container');

container.innerHTML = `<p style="color: red;">Error: ${message}</p>`;

}

fetch(apiUrl)

.then(response => {

if (!response.ok) {

throw new Error(`HTTP error! Status: ${response.status}`);

}

return response.json();

})

.then(data => {

renderUsers(data);

})

.catch(error => {

renderError(error.message);

});

**Using Templates for Better Organization**

For larger applications, use templates to organize the HTML structure:

function createUserTemplate(user) {

return `

<div class="user">

<h3>${user.name}</h3>

<p>Email: ${user.email}</p>

<p>Phone: ${user.phone}</p>

</div>

`;

}

function renderUsers(users) {

const container = document.getElementById('user-container');

container.innerHTML = users.map(user => createUserTemplate(user)).join('');

}

**Fetching and Rendering Example**

**Complete Code Example**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Dynamic Data Rendering</title>

<style>

.user {

margin: 10px 0;

padding: 10px;

border: 1px solid #ccc;

border-radius: 5px;

}

.user h3 {

margin: 0 0 5px;

}

.user p {

margin: 0;

}

</style>

</head>

<body>

<h1>Dynamic User List</h1>

<div id="user-container"></div>

<script>

const apiUrl = 'https://jsonplaceholder.typicode.com/users';

function renderUsers(users) {

const container = document.getElementById('user-container');

container.innerHTML = users.map(user => `

<div class="user">

<h3>${user.name}</h3>

<p>Email: ${user.email}</p>

<p>Phone: ${user.phone}</p>

</div>

`).join('');

}

function renderError(message) {

const container = document.getElementById('user-container');

container.innerHTML = `<p style="color: red;">Error: ${message}</p>`;

}

fetch(apiUrl)

.then(response => {

if (!response.ok) {

throw new Error(`HTTP error! Status: ${response.status}`);

}

return response.json();

})

.then(data => {

renderUsers(data);

})

.catch(error => {

renderError(error.message);

});

</script>

</body>

</html>

**Implementation Steps**

**a) Check Local Storage for Cached Data**

Before fetching new data, check if the data is already cached.

const cachedData = localStorage.getItem('apiData');

if (cachedData) {

console.log('Using cached data:', JSON.parse(cachedData));

}

**b) Fetch and Cache Data**

If no cached data exists, fetch it from the API and save it in Local Storage.

fetch('https://jsonplaceholder.typicode.com/posts/1')

.then(response => {

if (!response.ok) {

throw new Error(`HTTP error! Status: ${response.status}`);

}

return response.json();

})

.then(data => {

console.log('Fetched Data:', data);

localStorage.setItem('apiData', JSON.stringify(data)); // Cache data

})

.catch(error => {

console.error('Fetch Error:', error);

});

**c) Combining Cache Check and API Fetch**

Use cached data if available; otherwise, fetch from the API.

const cachedData = localStorage.getItem('apiData');

if (cachedData) {

console.log('Using cached data:', JSON.parse(cachedData));

} else {

fetch('https://jsonplaceholder.typicode.com/posts/1')

.then(response => {

if (!response.ok) {

throw new Error(`HTTP error! Status: ${response.status}`);

}

return response.json();

})

.then(data => {

console.log('Fetched Data:', data);

localStorage.setItem('apiData', JSON.stringify(data)); // Cache data

})

.catch(error => {

console.error('Fetch Error:', error);

});

}

**Enhancing Caching Logic**

**a) Expiry for Cached Data**

Add a timestamp to manage data expiration.

const cacheKey = 'apiData';

const expiryKey = 'cacheExpiry';

const cacheDuration = 60 \* 60 \* 1000; // 1 hour in milliseconds

const cachedData = localStorage.getItem(cacheKey);

const cacheExpiry = localStorage.getItem(expiryKey);

if (cachedData && cacheExpiry && Date.now() < cacheExpiry) {

console.log('Using valid cached data:', JSON.parse(cachedData));

} else {

fetch('https://jsonplaceholder.typicode.com/posts/1')

.then(response => {

if (!response.ok) {

throw new Error(`HTTP error! Status: ${response.status}`);

}

return response.json();

})

.then(data => {

console.log('Fetched Data:', data);

localStorage.setItem(cacheKey, JSON.stringify(data));

localStorage.setItem(expiryKey, Date.now() + cacheDuration); // Set new expiry

})

.catch(error => {

console.error('Fetch Error:', error);

});

}

**b) Clearing Cached Data**

Provide a mechanism to clear outdated or unnecessary cached data.

localStorage.removeItem('apiData');

localStorage.removeItem('cacheExpiry');

console.log('Cache cleared');

**Complete Example**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>API Data Caching</title>

</head>

<body>

<h1>API Data Caching with Local Storage</h1>

<div id="content"></div>

<script>

const cacheKey = 'apiData';

const expiryKey = 'cacheExpiry';

const cacheDuration = 60 \* 60 \* 1000; // 1 hour in milliseconds

const cachedData = localStorage.getItem(cacheKey);

const cacheExpiry = localStorage.getItem(expiryKey);

function renderData(data) {

const contentDiv = document.getElementById('content');

contentDiv.innerHTML = `

<h2>Post Title: ${data.title}</h2>

<p>${data.body}</p>

`;

}

if (cachedData && cacheExpiry && Date.now() < cacheExpiry) {

console.log('Using valid cached data');

renderData(JSON.parse(cachedData));

} else {

fetch('https://jsonplaceholder.typicode.com/posts/1')

.then(response => {

if (!response.ok) {

throw new Error(`HTTP error! Status: ${response.status}`);

}

return response.json();

})

.then(data => {

console.log('Fetched and caching data:', data);

renderData(data);

localStorage.setItem(cacheKey, JSON.stringify(data));

localStorage.setItem(expiryKey, Date.now() + cacheDuration);

})

.catch(error => {

console.error('Fetch Error:', error);

const contentDiv = document.getElementById('content');

contentDiv.innerHTML = '<p style="color: red;">Failed to fetch data. Please try again later.</p>';

});

}

</script>

</body>

</html>