Beginner’s Guide to Fetching & Displaying API Data with Local Storage

Subtitle: Learn how to fetch data from APIs, store it in the browser, and display it dynamically on your website.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Combining **APIs, Local Storage, and the DOM** allows you to create dynamic, interactive websites that retain data for faster load times. This guide shows beginners how to fetch, store, and display data efficiently.

Step 1: Fetch Data from an API

- Use JavaScript `fetch` to get data

```
fetch('https://jsonplaceholder.typicode.com/posts')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error(error));
```

- **Tip:** Always handle errors to prevent broken pages

Step 2: Store API Data in Local Storage

- Save the fetched data for later use

```
fetch('https://jsonplaceholder.typicode.com/posts')
  .then(res => res.json())
  .then(data => localStorage.setItem('posts', JSON.stringify(data)));
```

- **Takeaway:** Reduces repeated API calls and improves performance

---

Step 3: Retrieve Data from Local Storage

```
const posts = JSON.parse(localStorage.getItem('posts'));
console.log(posts);
```

- Check if data exists first to avoid fetching repeatedly

Step 4: Display Data on the Page

- Use DOM manipulation to render dynamic content

```
const container = document.querySelector('#postContainer');
posts.forEach(post => {
  const div = document.createElement('div');
  div.innerHTML = `<h3>${post.title}</h3><p>${post.body}</p>`;
  container.appendChild(div);
});
```

Step 5: Handling Refresh & Reload

- On page load, check if data exists in Local Storage
- If yes → display stored data
- If no → fetch from API and store it

```
window.addEventListener('DOMContentLoaded', () => {
  let posts = JSON.parse(localStorage.getItem('posts'));
  if (!posts) {
    fetchData(); // function to fetch and store data
  } else {
    displayPosts(posts); // function to render posts
  }
});
```

Step 6: Updating Data Dynamically

- Allow users to add or modify items
- Update Local Storage and refresh DOM elements

```
function addPost(title, body) {
  const posts = JSON.parse(localStorage.getItem('posts')) || [];
  posts.push({ title, body });
  localStorage.setItem('posts', JSON.stringify(posts));
  displayPosts(posts);
```

```
}
```

Step 7: Mini Project Idea

- **Dynamic Posts App:**
    1. Fetch posts from API
    2. Store posts in Local Storage
    3. Display posts dynamically
    4. Add, edit, delete posts with storage update
    5. Persist data on reload

Step 8: Best Practices

- Avoid storing large amounts of data (>5MB) in Local Storage
- Always parse JSON correctly
- Handle errors from failed API requests gracefully
- Keep code modular: separate fetching, storage, and display logic

---

Step 9: Combining With UI/UX

- Use cards, tables, or lists to display data cleanly
- Add search or filter functionality for better user experience
- Highlight newly added items dynamically

Step 10: Key Takeaways

- Fetch → Store → Display = core flow for dynamic web apps
- Local Storage improves performance and provides offline-like behavior
- Combining APIs + DOM + Storage enables interactive, persistent applications
- Practice by building small projects to master the workflow

Visit **haas.dev** for step-by-step tutorials, interactive project guides, and beginner-friendly JavaScript + Local Storage resources.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app
