Member-only story

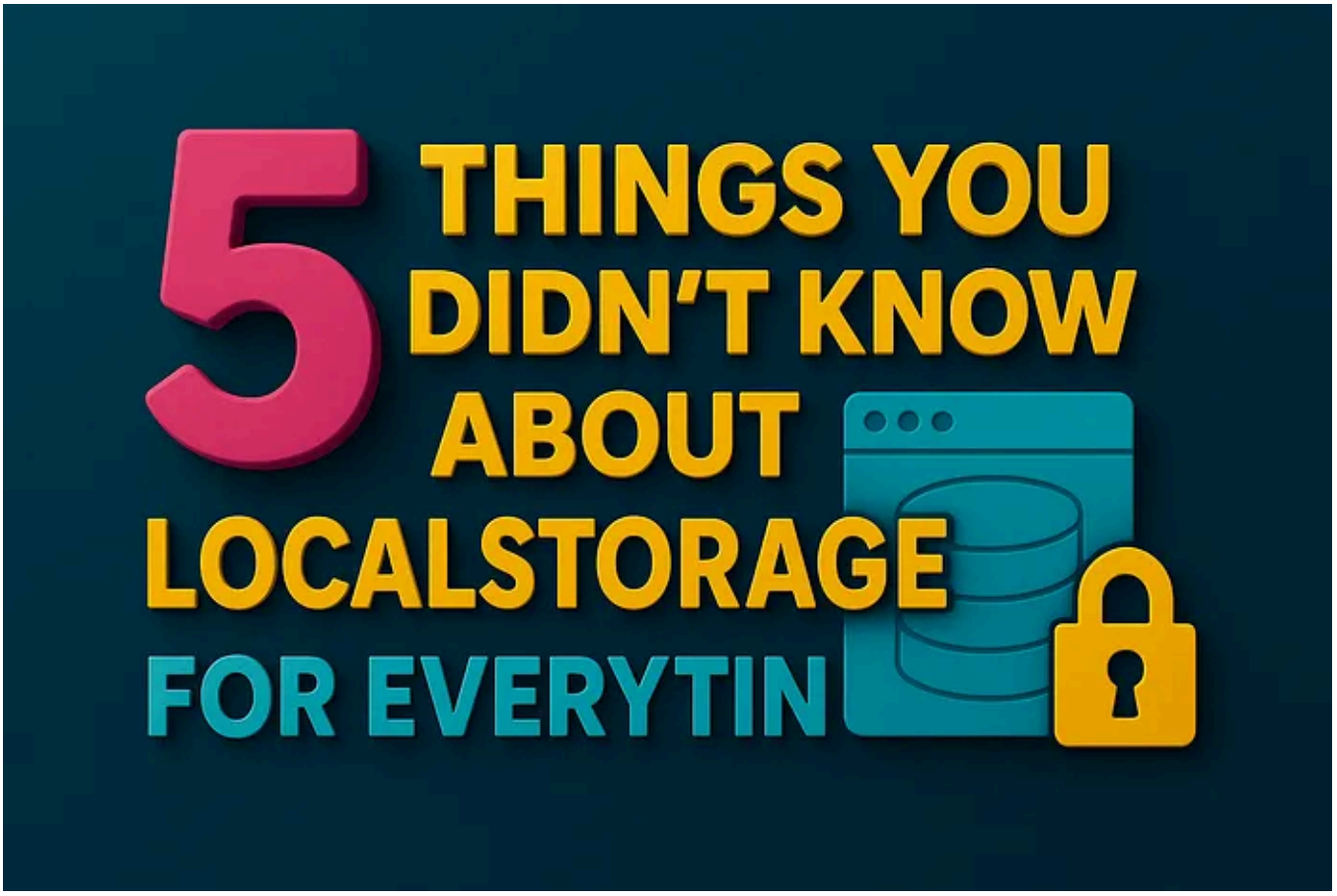
5 Things You Didn't Know About LocalStorage

Unlocking the Full Power of LocalStorage for Modern Web Apps

6 min read · 1 day ago

Deepak Mardi Follow

Listen Share More



Modern web development offers countless ways to store data — from REST APIs to cloud databases. But one of the simplest tools often overlooked is sitting right inside your browser: **LocalStorage**.

At first glance, LocalStorage seems basic: a simple key-value store for saving data in the browser. But under the hood, there are some powerful (and sometimes surprising) things you can do with it.

In this post, I'll reveal 5 things you probably didn't know about LocalStorage — and show how you can use it more effectively in your web projects.

Whether you're building a dashboard, a SaaS app, or a simple landing page, these tips will help you write better, faster, and more user-friendly code.

. . .

✨ What Is LocalStorage, Really?

Let's start with a quick refresher:

LocalStorage is part of the **Web Storage API** and allows developers to store data in the browser that persists even after the user closes the tab or browser window.

✅ It is **synchronous**
✅ It stores data as **string key-value pairs**
✅ The data persists **with no expiration** (until explicitly cleared)
✅ Capacity: **~5MB per origin** (depends on browser)

```
// Save data
localStorage.setItem('theme', 'dark');

// Retrieve data
const theme = localStorage.getItem('theme');
console.log(theme); // "dark"

// Remove data
localStorage.removeItem('theme');

// Clear all data
localStorage.clear();

// Store an object (must stringify)
const user = { name: 'Umar', age: 26 };
localStorage.setItem('user', JSON.stringify(user));

// Retrieve object (must parse)
```

```
const savedUser = JSON.parse(localStorage.getItem('user'));
console.log(savedUser); // { name: 'Umar', age: 26 }
```

Sounds simple, right? But there's much more going on under the hood. Let's dive in 👇

. . .

1️⃣ LocalStorage Isn't Shared Between Tabs — But It Can Be!

A common misconception is that LocalStorage is global to all browser tabs. Technically, each tab has its own in-memory **copy** of LocalStorage.

If you change a value in one tab:

```
localStorage.setItem('userStatus', 'loggedIn');
```

… it **won't** automatically update in other open tabs — unless you reload them.

But here's the trick 👇

You can listen to LocalStorage changes across tabs using the **storage event**:

```
// Listen for storage changes (in other tabs/windows)
window.addEventListener('storage', (event) => {
  console.log(`Key: ${event.key}`);
  console.log(`Old Value: ${event.oldValue}`);
  console.log(`New Value: ${event.newValue}`);
  console.log(`URL: ${event.url}`);
});
```

This allows you to **sync state across tabs** — super useful for apps where users might have multiple tabs open (think dashboards, chat apps, admin panels).

Use case:

- Log out users from all tabs
- Sync theme preferences across tabs
- Broadcast "new message" events across tabs

2️⃣ LocalStorage Works Without a Network — Perfect for Offline Apps

LocalStorage lives **entirely in the browser** — no network required.

This makes it ideal for building **offline-first** features:

- Caching user preferences
- Saving unsent form data
- Storing progress in long multi-step forms
- Offline “to-do” apps or note apps

For example:

```
// Save a post draft to localStorage
function saveDraft(post) {
  localStorage.setItem('draftPost', JSON.stringify(post));
}

// Load the post draft from localStorage
function loadDraft() {
  const draft = localStorage.getItem('draftPost');
  return draft ? JSON.parse(draft) : null;
}
```

```
// Example usage
const myPost = { title: 'My Blog Post', content: 'This is the content.' };
saveDraft(myPost);

const loadedPost = loadDraft();
console.log(loadedPost); // { title: 'My Blog Post', content: 'This is the cont
```

Now, even if the user refreshes the page, closes the tab, or loses their internet connection, their draft is safe.

. . .

3️⃣ You Can Store More Than Strings (Sort Of)

Yes — LocalStorage stores **strings**.

But with a little JSON magic, you can store **objects, arrays, booleans, numbers**:

```
// Define settings object
const settings = {
  darkMode: true,
  fontSize: 18,
  languages: ['en', 'fr', 'es']
};

// Save to localStorage (as a string)
localStorage.setItem('settings', JSON.stringify(settings));

// Later... load from localStorage and parse
const loadedSettings = JSON.parse(localStorage.getItem('settings'));

// Example usage
console.log(loadedSettings.darkMode);  // true
console.log(loadedSettings.fontSize);  // 18
console.log(loadedSettings.languages); // ['en', 'fr', 'es']
```

Tip: Always wrap your JSON.parse in a try/catch to avoid errors from corrupted data.

```
// Load settings from localStorage with error handling
function loadSettings() {
  try {
    return JSON.parse(localStorage.getItem('settings')) || {};
  } catch (e) {
    console.error('Failed to parse settings:', e);
    return {};
  }
}

// Example usage
const settings = loadSettings();
console.log(settings);
```

4️⃣ LocalStorage Has Limits — And They Matter

It's easy to assume "I can just store everything!" — but beware:

Browser limits:

- Chrome: ~5MB
- Firefox: ~10MB
- Safari: ~5MB

(Varies — use this browser storage limits guide)

If you exceed this, `localStorage.setItem()` will throw an error:

```
try {
  localStorage.setItem('bigData', bigValue);
} catch (e) {
  console.error('LocalStorage quota exceeded!', e);
}
```

Pro Tip:

Never store large binary data (images, videos) in LocalStorage. Use IndexedDB or Cache Storage for that.

5️⃣ LocalStorage Data Is Vulnerable — Handle It Carefully

This is critical: **LocalStorage is NOT secure.**

- Any JavaScript on the page can read it
- It is stored in plaintext on disk
- It is exposed to XSS attacks if your site is vulnerable

Never store:

- Passwords
- Auth tokens (better to use HttpOnly cookies)
- Sensitive personal data (PII)

If you *must* store semi-sensitive data:
✅ Encrypt it first

✅ Use a strong CSP (Content Security Policy)
✅ Sanitize user input to prevent XSS

Example:

```
import CryptoJS from 'crypto-js';

// Secret key (you should manage this securely!)
const SECRET_KEY = 'my_secret_key_123';

// Encrypt function
function encrypt(data) {
  const jsonData = JSON.stringify(data);
  return CryptoJS.AES.encrypt(jsonData, SECRET_KEY).toString();
}

// Decrypt function
function decrypt(cipherText) {
  const bytes = CryptoJS.AES.decrypt(cipherText, SECRET_KEY);
  const decryptedData = bytes.toString(CryptoJS.enc.Utf8);
  return JSON.parse(decryptedData);
}

// Example: save encrypted user profile
const userData = { name: 'Umar', email: 'umar@example.com' };
const encrypted = encrypt(userData);
localStorage.setItem('userProfile', encrypted);

// Example: load and decrypt user profile
const encryptedData = localStorage.getItem('userProfile');
if (encryptedData) {
  const decrypted = decrypt(encryptedData);
  console.log('Decrypted:', decrypted);
}
```

. . .

🚀 Real-World Use Cases for LocalStorage

Now that you know its power — here are some practical ways to use LocalStorage in your apps:

✅ Persist theme preference (`dark` vs `light`)
✅ Save UI state (collapsed menus, selected tabs)

✅ Store onboarding progress
✅ Cache non-critical API responses
✅ Offline drafts for forms
✅ Remember last search queries
✅ Save scroll position for long articles

🏆 Summary: Key Takeaways

Let's recap what you've learned:

✅ LocalStorage can sync across tabs using the `storage` event
✅ Perfect for offline-first features — no network required
✅ You can store objects & arrays using JSON
✅ It has size limits — don't abuse it
✅ It is not secure — never store sensitive data in it

LocalStorage is a fantastic **lightweight tool** for many use cases — but like any tool, you need to understand its strengths and limitations. Now you do!

. . .

🎉 Final Thoughts

Most devs think of LocalStorage as a basic API — but when used creatively, it can really improve the UX of your apps.

Armed with these tips, I hope you're inspired to leverage LocalStorage more effectively in your next project!

Programming | Web Development | JavaScript | Browsers | Front End Development

Follow

Written by Deepak Mardi

116 followers · 33 following

Software Developer | Unpacking JavaScript & React complexities into actionable insights.

No responses yet

Bgerby

What are your thoughts?

More from Deepak Mardi

Deepak Mardi

I Tried 40 Project/Note Management Apps... What I Chose, and My Top 10 List

By The Way, when I say note-taking, I include stuff like project management, ToDo & task management, class notes, etc...

Oct 8

Deepak Mardi

8 Apps to Use Instead of Doomscrolling on Your iPhone

These apps will actually be worth your time

5d ago

Deepak Mardi

Game theory is the cheat code to life

Game theory can explain humanity's biggest problem

Oct 9

Deepak Mardi

What Is The Best Diagramming Software in 2025

I wasted three years diagramming in Lucidchart before a sketchy looking napkin app doubled my team's output.

Oct 11

See all from Deepak Mardi

Recommended from Medium

In Stackademic by Somendradev

10 Niche Developer Tools You Didn’t Know Existed

Let’s be real—the developer world moves fast. Every week, a dozen new tools launch promising to “boost productivity” or “make your life...

Oct 15

Michal Malewicz

I was wrong about Liquid Glass.

It’s actually amazing—here’s why.

Oct 19

Will Lockett

You Have No Idea How Screwed OpenAI Actually Is

When you find yourself in a hole, at what point do you stop digging?

5d ago

Rahul Dinkar

Web Workers vs. WASM: When to Use Which for Performance

Web Workers vs WebAssembly (WASM): learn when to use each for performance. Keep UIs smooth, run faster code, and build next-gen apps.

Oct 16

In Coding Beauty by Tari Ibaba

Clean Code Is Dead

If you're still obsessed with writing "clean code" in 2025 then you are living in the stone age.

6d ago

Alexander Burgos

OKLCH: The Modern CSS Color Space You Should Be Using in 2025

Stop Fighting With Colors — Meet OKLCH

Oct 12

See more recommendations