SessionStorage

---

1. Core Mechanics & Lifecycle

- **Duration:** Data persists only for the duration of the **page session**.
- **The "Tab" Rule:** Unlike localStorage (which is origin-bound), sessionStorage is **tab-bound**.
- **The Survivor:** Data survives page reloads and restores (e.g., if the browser crashes and you restore the session).
- **The End:** Closing the specific tab or window deletes the data immediately.

---

2. Scoping: When are Tabs Shared? (The Exception)

This is a favorite interview "gotcha." Generally, tabs do **not** share sessionStorage. However, there is one specific scenario where data is copied:

- **Window.open():** If you use window.open('url') or a link with target="_blank", the new tab **initializes** by copying the entire sessionStorage from the parent tab.
- **The Catch:** Once the copy is made, the two tabs become **independent**. Updating a value in Tab A will **not** update it in Tab B.
- **Manual Opening:** If a user opens a new tab manually (clicking the "+" button) and types the URL, it starts with a **completely empty** sessionStorage.

---

3. Storage Limits & Performance

- **Capacity:** Same as localStorage (typically **~5MB** per origin).
- **Synchronous:** Just like localStorage, it blocks the main thread. Large reads/writes will cause UI jank.
- **Serialization:** Only stores strings. You must JSON.stringify and JSON.parse.

---

4. Error Handling & Exceptions

In a System Design interview, you must mention how you handle failures. sessionStorage can throw the following:

1. **QuotaExceededError (DOMException):** Thrown when you try to exceed the 5MB limit.
2. **SecurityError:** Thrown if the user has disabled cookies/storage or if you try to access it from an <iframe> with a different origin (sandboxing).

3. **TypeError:** Thrown if you try to use it in a non-browser environment (like SSR/Node.js) without a check.

---

5. Security: The XSS Vulnerability

Like localStorage, sessionStorage is fully accessible by any JavaScript running on the page.

- **XSS Risk:** If an attacker gets a script onto your page, they can read everything.
- **System Design Tip:** Do **not** store sensitive session tokens (like JWTs) here. Even though it dies when the tab closes, an active XSS attack happens while the tab is open.

---

6. Real-World Use Cases (The "Design" Part)

When would you choose this over other options?

- **Multi-step Forms:** Storing progress between Step 1 and Step 2 of a checkout. If the user opens the site in a *second* tab, you don't want the two checkouts to collide.
- **One-time Redirects:** Storing a returnTo URL after a login redirect.
- **Per-tab State:** Storing which filters are applied to a search result list, so the user can have two different searches open in two different tabs simultaneously.

---

7. Comparison: The Interview Summary Table

Feature	LocalStorage	SessionStorage	Cookies
Tab Isolation	Shared across all tabs	**Unique to each tab**	Shared across all tabs
Persistence	Permanent	**Until tab close**	Until expiry
Initial Copy	N/A (always there)	**Only on window.open**	N/A (always there)
Storage Event	Triggers in other tabs	**Never triggers** (it's isolated)	N/A

Most Asked Interview Q&A for SessionStorage

Q: "I have two tabs open for the same site. I update sessionStorage in Tab A. Does Tab B see it?"
A: No. sessionStorage is scoped to the top-level browsing context. They are isolated.

Q: "Can a Service Worker access sessionStorage?"
A: No. Service Workers do not have access to synchronous storage APIs like localStorage or sessionStorage. You would have to use IndexedDB or the Cache API inside a Service Worker.

Q: "How do you prevent 'Stale Data' in sessionStorage?"
A: Since sessionStorage clears on tab close, "stale data" is less of a long-term risk than localStorage. However, for a robust system, you should still implement a schema versioning check. If the app updates and the stored object format changes, the app should clear the old keys to avoid crashes.
