Comparison of *Web Workers* and *Shared Workers*

1. Introduction

JavaScript is single-threaded, meaning it can usually only do one thing at a time. Workers allow us to run scripts in the background (on a separate thread) so that the main web page remains responsive and doesn't freeze during heavy calculations or data fetching.

There are two main types of workers:

1. Dedicated Web Worker (usually just called "Web Worker")
2. Shared Worker

2. Comparison Summary

Feature	Dedicated Web Worker	Shared Worker
Accessibility (Scope)	Accessible only by the script that created it (one tab).	Accessible by multiple scripts, tabs, or iframes sharing the same domain.
Lifespan	Ends when the parent tab is closed.	Remains alive as long as at least one tab using it is open.
Communication	Direct communication via postMessage.	Communication via a port object. Requires port.start().
Complexity	Simple to implement.	More complex (requires managing ports).
Best Use Case	Heavy calculations for a single page.	Syncing data between tabs (e.g., chat apps, shopping carts).

3. Dedicated Web Worker

Definition

A Dedicated Worker is linked specifically to the browser tab that created it. If you open the same website in three different tabs, you will create three separate worker threads. They do not know about each other.

Key Characteristics

- Isolation: One worker per tab.
- Simple API: You send data using postMessage() and receive it using onmessage.

Code Example

main.js (The Page)

```
// Create the worker
const myWorker = new Worker("worker.js");

// Send data
myWorker.postMessage("Start Calculation");

// Receive data
myWorker.onmessage = function(e) {
  console.log("Result from worker:", e.data);
};
```

worker.js (The Background Script)

```
self.onmessage = function(e) {
  // Perform calculation
  let result = 100 * 5;
  self.postMessage(result);
};
```

4. Shared Worker

Definition

A Shared Worker is a script that can be accessed by multiple windows, tabs, or iframes, provided they come from the same origin (domain). If you open the website in three tabs, they all connect to the same single Shared Worker thread.

Key Characteristics

- Shared State: Data stored in the Shared Worker is available to all connected tabs.
- Ports: Communication happens through a "port". You must start the port to begin communication.
- Persistence: The worker stays alive until the *last* tab connected to it is closed.

Code Example

main.js (Tab 1 and Tab 2)

```
// Create (or connect to) the shared worker
const myShared = new SharedWorker("shared.js");

// Start the port (CRITICAL STEP)
myShared.port.start();

// Send data
myShared.port.postMessage("Hello from a Tab!");

// Receive data
myShared.port.onmessage = function(e) {
  console.log("Shared Worker says:", e.data);
};
```

shared.js (The Shared Background Script)

```
// Example: Tracking how many tabs are connected
let connections = 0;

self.onconnect = function(e) {
  const port = e.ports[0];
  connections++;

  port.start();
  port.onmessage = function(e) {
    // Broadcast or reply
    port.postMessage("You are connection number: " + connections);
  };
};
```

5. Conclusion

- Choose a Dedicated Worker if you simply need to offload a heavy task (like image processing or a big math loop) so the UI doesn't freeze. It is easier to write and debug.
- Choose a Shared Worker if you need different tabs to "talk" to each other (e.g., if a user logs out in one tab, the Shared Worker can tell the other tabs to log out too) or if you want to prevent downloading the same resource multiple times across different tabs.