**Web Worker:**

**What is web worker?**

* A web worker allows you to **run scripts in a separate background thread**.
* With this we can execute the heavy computations or long running tasks without blocking the main thread.

**How it works?**

* Web workers run in a **separate thread** from the main JavaScript execution thread.
* They don’t have direct access to DOM.
* **Communication** between Main thread & worker happens via **message passing** using **postMessage () & onmessage event listeners.**

**In Depth details:**

* Create Worker file in public folder – because Web Workers needs to be loaded via URL.
* To use Web Worker: In any react component or any other JS framework.

1. Create web worker.

*const newWorker = new Worker(“./worker.js”);*

1. Listen for messages.

*newWorker.onmessage = (event) => {};*

1. Send or post messages.

*newWorker.postMessage(“message”);*

1. Cleanup or terminate web workers.

*newWorker.terminate()*

* **Benefits:**

1. Avoids UI freezing – by running heavy calculations in the background
2. Non-blocking execution – main thread remains free for user interactions.
3. Improves performance – of the application.

* **Why Web Worker needs to be placed in public folder?**

1. Public folder is the place where assets are stored that should be publicly accessible.
2. Web Workers needs to be loaded using URL (that means not via an import statement).
3. Placing in public folder ensures that the React or any build process doesn’t interfere with the worker.
4. React’s or other build system processes files inside “src” folder, but web workers need to be referenced via URL.
5. Files in public are not bundled or transformed by the Webpack or any other bundlers.
6. Placing in public ensures it remains accessible at a fixed URL.
7. When we create a Web Worker using *new Worker (URL)*, the argument must be URL pointing to a separate file.
8. You cannot pass inline functions or import modules like normal JavaScript.
9. Webpack or any other bundler bundles all the “src” files together in to main.js file & worker.js cannot be accessed.
10. If *new Worker(“./worker.js”)* - tries to find an actual file, it will not find as it is bundled together.
11. If you still want to put it in “src” folder, you can use Blob URL or worker loader. They make sure it will not be bundled together with main.js and ensures to create as a separate file and so it can be accessible as an individual file.

* **How Web Worker file is secure, if it is kept as a separate file?**

1. By default, worker.js in public folder is not minified & is readable.
2. If you want to minify it, use webpack with worker-loader or manually with terser.
3. Using webpack, it ensures minification & separate worker execution.
4. Web Worker is served from the Same Origin (same domain & protocol) & so it follows the same origin policy (SOP) meaning:
5. Only your frontend app can access or interact with the worker.
6. External sites cannot directly access unless you allow explicitly CORS.

* **How Web Workers works BTS?**

1. JavaScript is single threaded, but Web Workers run in a separate thread provided by the browser.
2. Separate thread means:
   1. Separate execution context – Heap (own memory), Call Stack & Event Loop.
   2. Message system to communicate with main thread.
3. Web Workers are Browser APIs, not part of JavaScript.
4. They communicate asynchronously using message system, but they don’t use async/await.
5. They do not block the UI, making them useful for CPU intensive tasks.

* **Steps for execution flow of Web Workers:**

1. Main thread send message to start worker & the main thread continues.
2. Worker receives message & start a separate thread for execution.
3. Worker has its own execution context & event loop to execute Worker JS code in separate thread.
4. Once execution is done, worker sends message to Message queue & the event loop in the main thread handles the message to execute in main thread.
5. Message queue(workers) is part of Macro Task queue or task queue which handles timers, DOM APIs etc.
6. Communication between Worker & main thread is massage system.
7. Event loop in main thread handles workers message just like timers, DOM APIs etc. & put it to main thread call stack when it is free.

* **How multiple Web Workers execute?**

1. If there are multiple Web Workers, each one runs in separate thread.
2. Multiple workers mean multiple parallel threads runs independently.
3. Each worker/thread has its own Execution context & cannot share memory.
4. They communicate to main thread via postMessage ()– a messaging system and this is handled by browser’s internal message delivery system.
5. More workers mean better performance for CPU intensive tasks, but too many can overload.

* **What is Shared Worker?**

1. Shared Worker can be accessed by multiple scripts running in different windows, iframes, or different tabs as long as they are form the same origin.
2. Dedicated Workers are tied to single script where as Shared Workers enable inter-tab communication.
3. Examples of Real time use cases like Chat applications, live updates like stock market updates, cricket updates etc.
4. This kind of applications can use Shared Worker for each opened tab to get the data from Shared Worker & display the data instead of opening new connection or new dedicated Worker for each script.

* **What is Service Worker?**

1. Service Worker is a background script that acts as proxy between the browser and the network.
2. It enables features like offline support, caching, background sync and push notifications.
3. Example for Chat apps:
   1. If online messages are sent to the server immediately.
   2. If offline, messages are stored in Indexed DB & when the user comes online, the service worker sends the stored messages.
   3. The service worker registers sync event when message is saved offline and send messages o the server automatically when the network is available.

* **Push Notifications:**

1. Push notification is the message sent from the server to a user’s device even if the user is not actively using the app or website.
2. It appears as a Pop-up or alert on mobile devices, desktops even if the application is closed or running in the background.

* **How the Push Notifications work?**

1. First user grants permission to allow push notifications.
2. App registers/subscribes for push notifications service.
3. Server send the push notifications using Push API.
4. Service worker receives push notifications even if the app is closed.
5. The service worker wakes up and displays the message.
6. The browser wakes up the Service Worker to handle push notifications.
7. The Service Worker doesn’t run continuously, it only activates when a push event is received.

* **Message System:**

1. All types of Web Workers communicate via message system with main thread.
2. They cannot communicate with each other & they rely on main thread to communicate.
3. Workers cannot directly access to DOM or share memory.

* **Use cases of Web Workers:**

1. Data heavy computations like calculations etc.
2. Handling large JSON files like parsing large datasets.
3. Text analysis like spell checking etc.
4. Image processing in an editor etc.

* **Handle large datasets Efficiently in an application:**

1. Server-Side Pagination:
   * 1. Best for Large APIs because instead of fetching the entire dataset at once, handling the pagination on server.
     2. We can fetch only required number records per page.
2. Client-side pagination with Virtualization (In React):
3. If data must be fetched entirely & handle it in UI, use react Virtualized.
4. Virtualization can be done using react-window library.
5. Indexed DB for Persistent caching:
   * 1. Store API response in Indexed DB – a browser based database.
     2. Retrieve only required records from the DB instead of fetching data from the network every time.
6. Web Workers:
   * 1. Runs in background in a separate thread without blocking the UI.
     2. Good for CPU intensive tasks.

* **Summary of approaches:**

1. Web Workers: Good for heavy computations & background API calls.
2. Client-side pagination + Virtualization: Good for handling large datasets efficiently.
3. Indexed DB: Good for caching large API responses without blocking UI.
4. Best Approach (In React): Combine useState() + indexed Db to optimize performance.

**ES6 features:**

**Map & Set Data Structures:**

**JavaScript Questions & Answers:**