# Web Workers in WEB

CS-493: Enterprise Application Development


**Supervisor**
Mr. Atif Hussain

**Submitted By**

Amad Irfan 2021-CS-25

**University of Engineering and Technology**
**Lahore, Pakistan**

# Contents

# 1 Web Workers

## 1.1 what is Web Workers?

They are scripts that run in the background, separate from the main thread of your website. The main purpose of Web Workers is to enable concurrent processing and improve performance by offloading computationally intensive tasks from the main thread. Data is sent between workers and the main thread via a system of messages — both sides send their messages using the postMessage() method, and respond to messages via the onmessage event handler (the message is contained within the message event's data attribute).
When executing scripts in an HTML page, the page becomes unresponsive until the script is finished.
A web worker is a JavaScript that runs in the background, independently of other scripts, without affecting the performance of the page. You can continue to do whatever you want: clicking, selecting things, etc., while the web worker runs in the background.

## 1.2 Advanced use cases of Web Workers.

- **Image Processing:**
  **Scenario:** Image processing tasks, such as resizing, filtering, or applying complex algorithms, can be computationally expensive and may cause the main thread to freeze, leading to a poor user experience.
  **Use of Web Workers**: By offloading image processing tasks to Web Workers, these operations can be performed in the background without affecting the main thread. This allows for a smoother user interface, especially when dealing with large images or a high volume of image processing tasks.

- **Data Fetching and Parsing:**
  **Scenario:** Fetching and parsing large datasets from APIs or external sources can be time-consuming. If done on the main thread, it can result in UI freezes and delayed user interactions. **Use of Web Workers:** Web Workers can handle data fetching and parsing asynchronously. This prevents the main thread from being blocked, ensuring that the user interface remains responsive while the worker thread handles the data-related tasks. This is particularly useful in applications dealing with real-time data updates.

## 1.3 Future use of web workers.

- **Enhanced Real-Time Collaboration:**
  Web Workers could be employed to facilitate real-time collaboration in web applications, allowing multiple users to work on shared documents or projects simultaneously without affecting each other's user interfaces.

- **Augmented and Virtual Reality:**
  As augmented reality (AR) and virtual reality (VR) become more prevalent on the web, Web Workers could be utilized to handle the complex computations required for rendering 3D environments, simulations, and immersive experiences, ensuring a smooth and responsive user experience.

- **Edge Computing Integration:**
  With the rise of edge computing, Web Workers could play a role in offloading certain computations to edge devices, reducing latency and enhancing performance for applications that require real-time responsiveness.

# 2 How web Workers works in web:

## 2.1 Creation:

- In a web application, the developer decides to use a Web Worker for a specific task, such as data processing or complex calculations.
- A separate JavaScript file is created to contain the code for the Web Worker. This file operates independently of the main thread.

## 2.2 Instantiation:

- In the main thread of the web application, a new instance of the `Worker` object is created, specifying the URL of the Web Worker script. This URL can be the path to the JavaScript file created in the first step.

## 2.3 Communication Channel:

- The main thread and the Web Worker communicate through a message-passing mechanism. The main thread and the Web Worker can send messages to each other using the `postMessage` method.

## 2.4 Task Execution:

- The Web Worker script contains the logic for the task that needs to be performed in the background. This script runs independently in its own thread, separate from the main thread.

## 2.5 Message Handling:

- The Web Worker script has an `onmessage` event handler that listens for messages from the main thread. When a message is received, the script processes the data and may perform complex computations.

## 2.6 Result Posting:

- After completing the task, the Web Worker uses the `postMessage` method to send the result or any relevant information back to the main thread.

## 2.7 Main Thread Response:

- The main thread has an `onmessage` event handler as well, which listens for messages from the Web Worker. When the message is received, the main thread processes the result and takes appropriate actions, updating the user interface or performing other tasks as needed.

## 2.8 Isolation and Termination:

- Web Workers operate in a separate global context, meaning they don't share variables or resources with the main thread. This isolation helps prevent conflicts.
- When the Web Worker is no longer needed, it can be terminated using the `terminate` method.

# 3 Limitations or Disadvantages of Web Workers

## 3.1 No Access to the DOM:

Web Workers operate in a separate thread with no direct access to the Document Object Model (DOM). This means they cannot manipulate the DOM directly, limiting their ability to interact with the user interface.

## 3.2 Communication Overhead:

Communication between the main thread and Web Workers is achieved through message passing. Passing large amounts of data between threads can incur overhead, potentially offsetting the performance gains achieved by using Web Workers.

## 3.3 Limited Synchronous Execution:

Web Workers primarily operate asynchronously, handling tasks in the background. However, certain operations, such as synchronous XMLHttpRequests, are not allowed within a Web Worker, limiting their capabilities in certain scenarios.

## 3.4 Browser Compatibility:

While Web Workers are supported in modern browsers, older browsers or specific environments may not fully support them. Developers need to check for compatibility and provide fallbacks or alternative strategies if necessary.

## 3.5 Increased Memory Usage:

Each Web Worker instance consumes additional memory. In scenarios where a large number of workers are created, this can lead to increased memory usage and potential performance issues.

# 4 Links

- live app link : `https://webworkerai.netlify.app/`
- Github link : `https://github.com/AmadIrfan/WEB-WORKERS-IN-JS.git`