IUT GON – MMI – S5 – Dev Front TD 2 – *Web Workers*

L'API JavaScript *WebWorker* permet d'exécuter plusieurs tâches simultanément sous la forme de scripts indépendants. Ceci est basé sur les notions de concurrence et de parallélisme.

Parallélisme : plusieurs tâches s'exécutent en même temps en utilisant plusieurs cœurs du processeur (CPU). Néanmoins, le parallélisme est plutôt une spécification du système, c'est le planificateur de tâches qui décide d'exécuter des *threads* sur des cœurs différents.

Lorsque la Machine Virtuelle (VM) JavaScript rend une page et des composants de l'interface utilisateur sur son fil d'exécution, une instance de *WebWorker* lance son *thread* sur un noyau distinct, en arrière-plan, permettant d'exécuter plusieurs tâches à la fois (interactions , calculs, …).

Concurrence : alterner l'exécution de plusieurs tâches sur un même cœur. Le passage d'une tâche à une autre est très rapide, imperceptible.

Utilisation des *WebWorkers* : tâches longues à s'exécuter (réception/transmission, chargement, codage/décodage de certaines données), tâche de fond.

1) Création et Communication *WebWorker* – application principale

La création d'un WebWorker s'effectue avec :

```
const worker = new Worker("worker.js");
```

où le document `worker.js` contient le code que le *WebWorker* exécute.

La méthode `postMessage()` permet d'envoyer un message :

- Depuis l'app principale vers le *WebWorker* : `worker.postMessage("coucou")`
- Depuis le *WebWorker* vers l'app principale : `postMessage("worker coucou")`

L'événement `message` (ou `onmessage()`) permet de réceptionner un message :

- Dans l'app principale :

```
worker.addEventListener("message", (e) => {
      console.log(e.data);  // les données transmises
},false)
```

- Dans le *WebWorker* :

```
self.addEventListener("message", (e) => {
      console.log(e.data);  // les données transmises
},false)
```

Exercice 1. Considérez le document exo1.html et exo1-worker.js. Testez le comportement dans le navigateur en ouvrant la console.

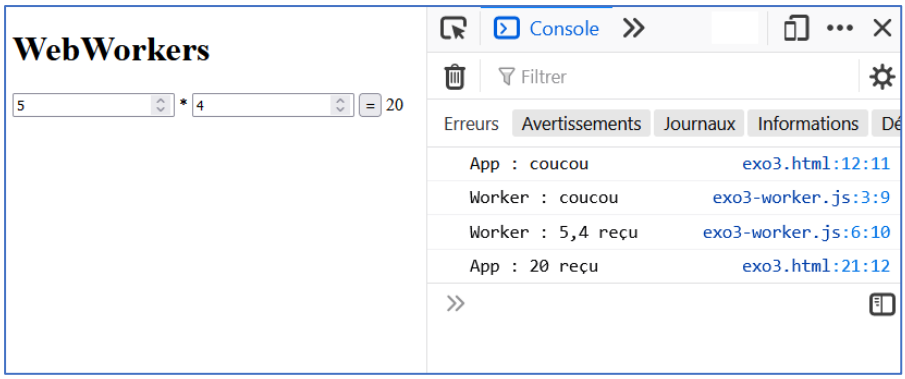
2) Stopper un *WebWorker*

- Dans l'app principale : `worker.terminate()`

- Dans le *WebWorker* : `self.close()`

Exercice 2. Complétez le code de l'exercice 1 pour que le *WebWorker* se termine après avoir envoyé son message, reçu celui de l'app et envoyé en retour « bien reçu ».

Exercice 3. L'app permet de saisir deux nombres (exo3.html), les transmet à un *WebWorker* (sous la forme d'un tableau) qui les multiplie et transmet le résultat en retour à l'app. Complétez le document HTML et le document JS du *WebWorker* pour que les nombres saisis soient transmis lorsque l'U clique sur l'élément <button>, et que le résultat de la multiplication envoyé par le *WebWorker* soit affiché via l'élément <output> du document HTML. Notez que vous devez obtenir l'affiche ci-dessous dans la console (initialisation + réception messages).



L'exercice 3 n'est qu'un exemple type, bien entendu dans le cas d'une calculatrice basique, il est inutile de faire appel à l'API *WebWorker*.

3) Cas d'utilisation : calcul plus ou moins long

Nous prenons l'exemple du calcul de la somme des n premiers entiers pour n grand, codé dans le document calcul-long.html. Constatez que le contenu de la page est inaccessible tant que le calcul n'a pas abouti. Le pire serait donc une boucle infinie (soyez patients si vous testez).

Existe-t-il une solution en utilisant la programmation asynchrone ?

Exercice 4. L'objectif consiste à faire effectuer le calcul par un *WebWorker*. L'U peut :

- Saisir un nombre (100000000 par défaut), qui sera nommé `nbUpTo` en JS.
- Démarrer le calcul lorsque qu'il clique sur un bouton « calculer » (accessible seulement si le calcul n'est pas en cours). L'app envoie le nombre au *WebWorker*. Avant de démarrer le calcul, le *WebWorker* affiche dans la console « Worker : calcul lancé » et envoie le message « start » à l'app qui affiche dans la console « App : calcul lancé ».
- Terminer le *WebWorker* depuis l'app avec un bouton « terminer », et donc stopper le calcul. Tous les boutons deviennent inaccessibles.

Lorsque le calcul aboutit, son résultat est transmis à l'app avec un message. L'app termine le *WebWorker* et l'affiche dans la page.

Un même élément HTML est utilisé pour afficher tous les messages reçus dans l'app.

Les messages auront la structure suivante :

- Démarrer le calcul depuis l'app : `{ "cmd": "start", "upto": nbUpTo }`
- Envoyer un message start depuis le *Worker* : `{ "cmd": "start" }`
- Retourner le calcul depuis le *Worker* : `{ "cmd": "result", "result": result}`

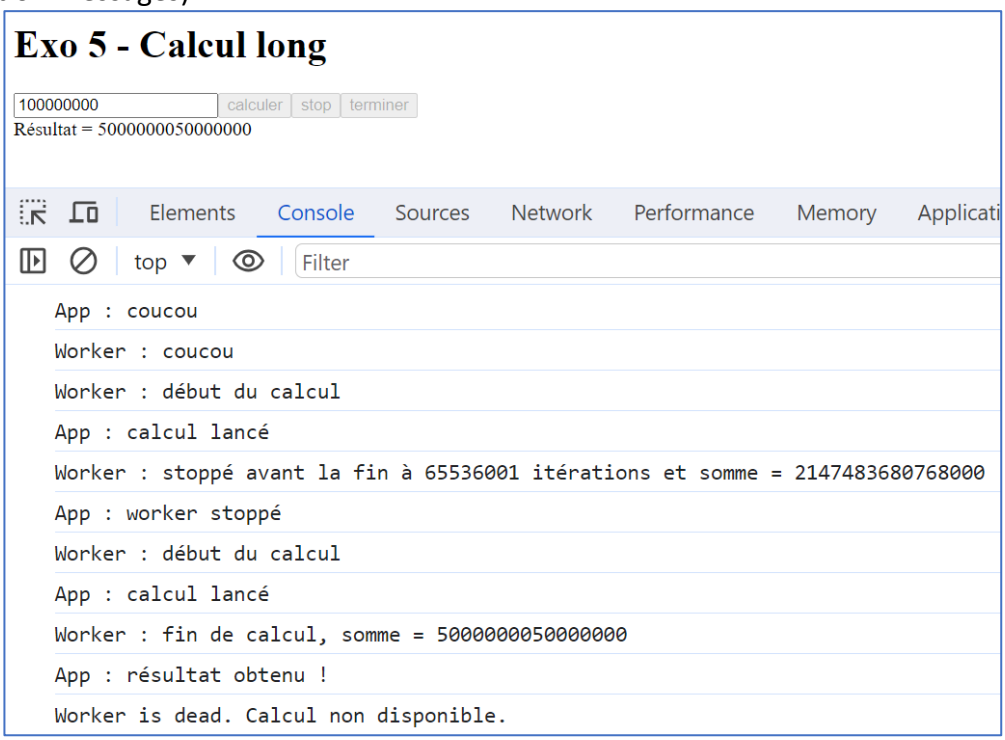
Notez qu'ils sont basés sur la transmission de commandes.

Exercice 5. Sans utiliser de *WebWorker*, proposez une solution pour stopper le calcul (sur clic sur un bouton « stopper ». Le calcul pourra être repris une fois le calcul stoppé ou terminé.

Exercice 6. En vous servant de l'exercice 5, compléter/modifiez le code l'exercice 4 pour que l'U puisse stopper le calcul sans terminer le *Worker* :

- Stopper le calcul en cours avec un bouton « stop » (accessible seulement si le calcul est en cours). Le *Worker* stopper le calcul et envoie un message « stop » à l'app qui affiche dans la page « App : calcul stoppé ». Le calcul peut être démarré à nouveau avec le bouton « calculer ». La structure du message de l'App vers le Worker pour stopper le calcul est la suivante : `{ "cmd": "stop" }`

Notez que vous devez obtenir l'affiche ci-dessous dans la console (initialisation + réception messages).



4) Cas d'utilisation : récupération de données auprès d'un serveur

Un 2ème cas d'utilisation classique d'un WebWorker est la récupération de données auprès d'un serveur (avec la méthode `fetch()`).

Exercice 7. L'objectif consiste à créer un *WebWorker* qui récupère des données de poésie depuis https://poetrydb.org/ et les transmets à l'app pour affichage dans la page. L'U clique sur un bouton pour demander l'affichage de N poésies (il peut contrôler ce nombre N). L'app envoie un message indiquant que le *Worker* doit récupérer un nombre N de poésies sélectionnées aléatoirement dans le corpus. Les poésies sont

transmises une par une à l'app, qui les affiche au fur et à mesure de leur réception. L'affichage d'une poésie comprend son titre, son auteur et le contenu de la poésie (les lignes). Lorsque toutes les poésies demandées ont été transmises à l'app, le *Worker* indique à l'app que la transmission est terminée. L'U peut demander d'afficher d'autres poésies en cliquant de nouveau sur le bouton.