PHP Worker Threads

- Initiating web workers
- Controlling web workers

PHP is always ran in a web worker to ensure the PHP runtime doesn't slow down the user interface of the main website.

Imagine the following code:

```
<button onclick="for(let i=0;i<100000000;i++>) {}">Freeze the page</button>
<input type="text" />
```

As soon as you click that button the browser will freeze and you won't be able to type in the input. That's just how browsers work. Whether it's a for loop or a PHP server, running intensive tasks slows down the user interface.

Initiating web workers

Web workers are separate programs that can process heavy tasks outside of the main application. They must be initiated by the main JavaScript program living in the browser tab. Here's how:

```
const phpClient = consumeAPI<PHPClient>(
    spawnPHPWorkerThread(
        '/worker-thread.js' // Valid Worker script URL
    )
);
await phpClient.isReady();
await phpClient.run({ code: `<?php echo "Hello from the thread!";` });
```

Controlling web workers

Exchanging messages is the only way to control web workers. The main application has no access to functions or variables inside of a web worker. It can only send and receive messages using

`worker.postMessage` and `worker.onmessage = function(msg) { }`.

This can be tedious, which is why Playground provides a convenient consumeAPI function that abstracts the message exchange and exposes specific functions from the web worker. This is why we can call `phpClient.run` in the example above.

□□□ #2
□ □□□ □□□ 2026-08-29 15:36:35 UTC
□ □□□ □□□ 2026-08-29 15:36:35 UTC