Unit 2: HTML5, JQuery and Ajax

HTML5 Geolocation API:

The Geolocation API of HTML5 helps in identifying the user's location, which can be used to provide location specific information or route navigation details to the use. There are many techniques used to identify the location of the user. The Geolocation API protects the user's privacy by mandating that the user permission should be sought and obtained before sending the location information of the user to any website. So the user will be prompted with a popover or dialog requesting for the user's permission to share the location information. The user can accept or deny the request.

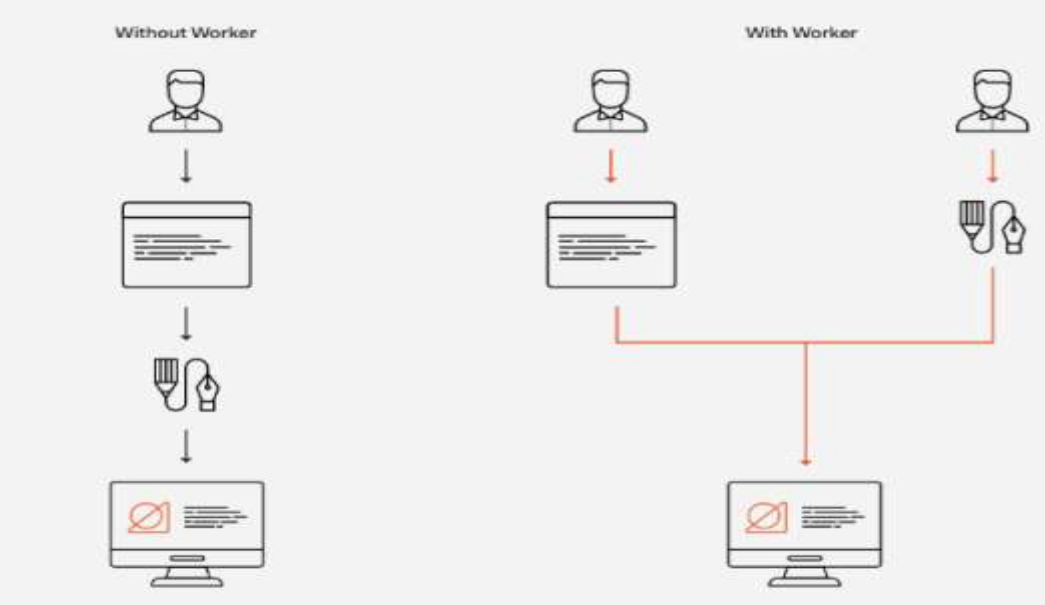
The current location of the user can be obtained using the getCurrentPosition function of the navigator.geolocation object. This function accepts three parameters – Success callback function, Error callback function and position options. If the location data is fetched successfully, the success callback function will be invoked with the obtained position object as its input parameter. Otherwise, the error callback function will be invoked with the error object as its input parameter.

Web Workers

Web Workers are a simple means for web content to run scripts in background threads.The worker thread can perform tasks without interfering with the user interface. In addition, they can perform I/O using XMLHttpRequest(although the responseXML and channel attributes are always null).Once created, a worker can send messages to the JavaScript code that created it by posting messages to an event handler specified by that code (and vice versa).

Using web workers in HTML5 allows you to prevent the execution of bigger tasks from freezing up your web page. A web worker performs the job in the

background, independent of other scripts and thus not affecting their performance. The process is also called threading, i.e., separating the tasks into multiple parallel threads. During the time, the user can browse normally, as the page stays fully responsive.



TYPES OF WEB WORKERS

Dedicated Workers

- Dedicated Web Workers are instantiated by the main process and can only communicate with it.
- A dedicated worker is only accessible by the script that called it.

Shared Workers

- Shared workers can be reached by all processes running on the same origin (different browser tabs, iframes or other shared workers).
- A shared worker is accessible by multiple scripts, similar to the basic

dedicated worker, except that it has two functions available handled by different script files

Workers are created and executed in one of the main program files with their code housed in a separate file. A worker is built using the Worker constructor method, which takes a parameter of worker.js, the JavaScript file storing the worker code.

Using .postMessage(), the parent thread can communicate messages to its workers. .postMessage() is a cross-origin API the can transmit primitive data and JSON structures, but not functions.

The parent code may also have a callback function that listens for a response from the worker confirming its work is complete in order for it to enact another action. As in the example below, the callback function will contain a target, which identifies the worker ('message') , and data, or the message posted by the worker.

This is main.js.

```
let worker = new Worker('worker.js');

worker.postMessage("Hello World");

worker.addEventListener('message', function(e) {
  console.log('Worker said: ', e.data);
}, false);
```

This is worker.js

```
self.addEventListener('message', function(e) {
  self.postMessage(e.data);
}, false);
```

Meanwhile, the worker, living in its own file, stands by with an eventListener waiting to be called. When it receives the message event from the parent code, it likewise communicates its response via the postMessage method.

HTML5 File API (Other Features in HTML5 – for reference only)

The HTML5 file API enables JavaScript inside HTML5 pages to load and process files from the local file system. Via the HTML5 file API it is possible for JavaScript to process a file locally, e.g. compress, encode or encrypt it, or upload the file in smaller chunks. Of course the HTML5 file API raises some security concerns.

Core Objects of HTML5 File API

The HTML5 file API contains the following core objects:

- FileList
- File
- Blob
- FileReader

The File object represents a file in the local file system.

The FileList object represents a list of files in the local file system. For instance, a list of files inside a directory.

The Blob object represents a Binary Large OBject (BLOB) which is used to hold the contents of a single file from the local file system.