
# JavaScript - Lesson 5
# WebSockets, Workers & co.

# Intervenant

- Quentin Pré
  <pre.quentin@gmail.com>

- MTI 2014


- Lead Front-End Engineer
  @ Adikteev / MotionLead


- I make digital ads for a living,


- I’m pretty sure there is a
  special place in hell for this.

# WebSockets

# What are Websockets ?

- A response to HTTP being slow for emerging network intensive applications.

- A persistent connection over TCP between the client and the server.

- Event driven.

Spec: https://tools.ietf.org/html/rfc6455

# API

WebSocket: main object for instantiating a connection and interacting with it.

MessageEvent: an interface representing the messages received over WebSocket.

CloseEvent: sent to the client when the connection is closed.

# WebSocket()

**instantiate**:

*new WebSocket(url)*, where url starts with “ws://” the protocol for websockets.

**methods**:

- send(): sends a message over the connection (can be of type String, ArrayBuffer of Blob)

- close(): closes the connection

# Availability

- Front-end: WebSockets are available on every major platform now.
- Backend: To use WebSockets you will need to install one flavour of your choice through npm:
  - The most basic is WS
  - SocketIO was historically the most advanced implementation on both server side and client side, it's been caught up by the crowd now.
  - …

# Server

```
const WebSocket = require('ws');

const WSServer = new WebSocket.Server({
  port: process.env.port,
});

const onConnection = (client) => {
  client.send('Hello client !');
}

WSServer
  .on('connection', ws => onConnection(ws));
```

# Usage with Express

```
const express = require('express');
const WebSocket = require('ws');
const http = require('http');

const app = express();
const HTTPServer = http.createServer(app);
const WSServer = new WebSocket.Server({
  server: HTTPServer
});

WSServer
  .on('connection', ws => onConnection(ws));

HTTPServer.listen(process.env.port);
```

- WS can plug into an existing HTTP server if needed

# Client

```
const ws = new WebSocket('ws://...');

const onOpen = () => {
  ws.addEventListener('message', onMessage);
};

const onMessage = (message: MessageEvent) => {
  console.log(message.data);
}

ws.addEventListener('open', onOpen);
```

*WebSocket* is a native object in modern navigators

# Workers

- There are two kinds of Workers:
  - **WebWorkers**: allowing you to offload intensive computations to another VM
  - **ServiceWorkers**: Acting as proxy servers between your app and the network layer.

# WebWorker

**instantiate**:
*new Worker(url)*, where url points to the script to be executed.

**methods**:
- postMessage(): sends a message to the worker.
- terminate(): kills the Worker's VM.

**caveats:**

- A WebWorker does not have access to the DOM
- A WebWorker leverages the whole power of a new V8 instance, and therefore its consumption too…
- **SharedArrayBuffer** support is still sparse, meaning that you have to serialise and deserialise your results between instances, which is VERY expansive.

# ServiceWorkers

- Acting as proxy servers between your app and the network layer.
- Downloaded every 24hours
- Installed if changed
- Used for:
    - Background data sync
    - Offline websites
    - more efficient caching strategy (avoid http roundtrip)
    - push notifications
- Extensive Documentation : https://developer.mozilla.org/en/docs/Web/API/Service_Worker_API
- Google intro: https://developers.google.com/web/fundamentals/getting-started/primers/service-workers

# ServiceWorker

**instantiate:**
*ServiceWorkerContainer.register(scriptURL, options)*
, where url points to the script to be executed.

**caveats:**
-Subject to CORS Restrictions (third party scripts can not use it for caching)

**bonus:**
- WebPack has a plugin to automate the creation of a caching ServiceWorker: **SWPrecacheWebpackPlugin**

# Progressive Web Apps

A progressive web app is an application that uses a few different techniques to lower its footprint:

- Is minified (duh !)
- Uses Immutable Assets to optimise caching
- Uses a ServiceWorker for caching network requests
- Uses Immutable Assets
- Uses Dynamic imports
- Uses HTTP/2
- …

check if your app follows the guidelines via Google Lighthouse !

# Progressive Web Apps

É que s’appelleriou Quezac ?

See Google’s PWA guidelines and tools

# Reading

Not JS related, but a basic of software engineering:
Out Of the Tar Pit

Have a question later ?

<pre.quentin+mti2019@gmail.com>

“Gouter, on va gouter !”

– Richard Bullit

Get the workshop assets here:
https://github.com/qpre/tp4-mti-websockets

# New rules

- limitation 140 carateres
- page profil utilisateur
- recherche (pas algolia)
- gestion de fichier pieces jointes