Introduction

**But du talk :** Développer une web app cross platform qui permet d’interagir entre PC et mobile.

**Qu’est-ce que GRUNT :** Grunt est simplement un *JavaScript Task Runner*, un outil vous permettant de créer des tâches automatisées en JavaScript avant ou pendant le runtime.

<http://gruntjs.com/>

**Qu’est-ce que NodeJS :** Node.js est avant tout la possibilité d'exécuter du JavaScript hors du contexte du navigateur, en général pour y faire un serveur web. Il est populaire du fait de sa vitesse d'exécution et de son aspect asynchrone.

<http://nodejs.org/>

**Qu’est-ce que Socket.io :** Socket.IO permet de faire la liaison entre vous et les autres personnes connectées sur la même application Web, le tout en temps réel via des websocket.

<http://socket.io/>

La connection cross platform

* Récupérer template à l’url suivante : <http://we.tl/Gsg8gdBMQZ>
* Placer le dossier dans son dossier wamp (utilisé comme point d’accès)
* Récupérer son adresse IP (*cmd* puis “*ipconfig*”).
* La mettre dans le template *desktop.html* et *mobile.html* + port 1337
* Lancer wamp → put online
* Sur desktop → <http://localhost/webapp> et sur mobile → [http://IP/webapp](http://ip/webapp)
* Si ça ne fonctionne pas éditer httpd.conf → allow from all
* SI ça ne fonctionne toujours pas → tant pis. Deuxième onglet avec url direct sur <http://localhost/webapp/mobile.html>

Installation de NodeJS et configuration

* Installer nodejs et penser à l’inclure dans le path
* Lancer la console (*cmd*) puis se rendre dans le répertoire de la webapp (*cd c:/wamp/www/webapp\_template*)
* Lancer la commande suivante. Elle va installer tous les modules nécessaire à notre webapp tout en les enregistrant dans le fichier *package.config (--save).* Dans notre cas on utilise que des plugins grunt car on souhaite l’exploiter à fond. Tous les autres plugins sont disponible ici <http://gruntjs.com/plugins> et beaucoup sont très intéressant.

*npm install --save grunt --save grunt-express@1.4.0 --save grunt-contrib-uglify --save grunt-contrib-cssmin --save grunt-contrib-jshint --save grunt-contrib-watch --save grunt-connect-socket.io --save express --save socket.io*

Configuration de Grunt

* Éditer le fichier *Gruntfile.js*
* On a besoin de charger les modules dans notre application. C’est les mêmes noms que les dossiers que vous pouvez voir dans *node\_modules*. Pour cela :

**grunt.loadNpmTasks('grunt-contrib-uglify'); // Minifier/Concaténer les fichiers JS**

**grunt.loadNpmTasks('grunt-contrib-cssmin'); // Minifier/Concaténer les fichier CSS**

**grunt.loadNpmTasks('grunt-contrib-jshint'); // Compilateur JS**

**grunt.loadNpmTasks('grunt-contrib-watch'); // Watcher d’événement**

**grunt.loadNpmTasks('grunt-connect-socket.io'); // Socket.IO pour grunt**

**grunt.loadNpmTasks('grunt-express'); // NodeJS pour grunt**

* Ensuite il faut préciser à grunt qu’on souhaite les utiliser dans une certaine configuration. Le “!” permet d’exclure certains fichier. Les expressions régulières fonctionnent pour les noms de fichiers. A savoir, si des tableaux sont récurant, il est possible d’en faire des variables (c’est un fichier JS ;) ).

jshint: {

all: ['Gruntfile.js', ‘js/\*.js', ‘!js/\_built.js’]

},

uglify: {

options: {

separator: ';',

mangle: false

},

compile: {

src: ['js/\*.js', ‘!js/\_built.js’],

dest: ‘js/\_built.js’

}

}

watch: {

scripts: {

files: ['Gruntfile.js', ‘js/\*.js’, ‘!js/\_built.js’],

tasks: [‘scripts’]

}

},

express: {

webappserver: {

options: {

server: path.resolve(‘js/server’),

keepalive: true,

port: 1337,

hostname: ‘localhost’,

showStack: true,

}

}

}

* Une fois que tout ça est fait, il faut appeler les tâches que l’on souhaite exécuter

grunt.registerTask('default', ['scripts', 'watch']);

grunt.registerTask('scripts:dev', ['jshint', 'uglify:compile', 'cssmin:compile']);

grunt.registerTask('roomServer', ['express', 'express-keepalive']);

* Ainsi, on peut désormais lancer dans son terminal la commande “*grunt*” afin d’éxecuter les tâches effectuer par le watcher (uglify, jshint...etc.). Ou lancer la commande “*grunt webappserver*” pour lancer le serveur nodejs express.

Débutons avec NodeJS et Socket.io

* Éditer le fichier “*server.js*”. ça sera, comme son nom l’indique, notre serveur node. On a besoin d’inclure certaines librairie node pour l’initialisation d’un serveur utilisant socket.io.

var express = require('express');

var http = require('http');

var app = express();

var server = http.createServer(app).listen(1337);

var io = require('socket.io').listen(server);

exports = module.exports = server;

// delegates use() function

exports.use = function() {

app.use.apply(app, arguments);

};

* Maintenant on est prêt à ouvrir le dialogue entre device ! Socket.io fonctionne par événement. Il est simplement capable d’en écouter ou d’en emmètre. Il possède aussi quelques évènement natif comme la connexion ou la déconnexion. Testons cela :

io.sockets.on('connection', function(socket) {

console.log('Nouvel utilisateur');

socket.on('disconnect', function(reason) {

console.log('L\'utilisateur s\'est déconnecté.');

console.log(reason);

});

});

* Il y a plusieurs façon d'émettre un événement. *io.sockets.emit* émettra un évènement à tous les utilisateurs connectés tandis que *socket.emit* lui émettra un événement seulement à l’utilisateur courant (ayant déclenché le dernier événement). Enfin, *socket.broadcast.emit* lui émettra un événement à tous les utilisateurs sauf l’utilisateur courant.
* Il ne nous reste plus qu’à gérer la connexion à socket.io côté client en éditant *client.js* :

var socket = io.connect('http://IP:PORT');

* Essayer de se connecter via desktop ET via mobile. Regarder la console et voir l’affichage de nouveaux utilisateurs

**LE PROJET EST PRÊT ET OPÉRATIONNEL**

Premiers pas avec Socket.IO

On va faire avancer la balle du desktop en appuyant sur le bouton de la vue mobile. Pour ça, on bind un événement au clique sur le bouton et on l’envoie au server :

$(document).ready(function(){

$('.button').click(function(){

socket.emit('goBall', { distance: 30 } );

});

});

L’événement est émit. Maintenant il faut le catcher au niveau du serveur pour l’envoyer au desktop

socket.on('goBall', function(params){

socket.broadcast.emit('goBallViaServer', params);

});

Enfin, côté desktop, il faut catcher l’event et agir en conséquence

socket.on('goBallViaServer', function(params){

$('.ball').animate({left: ($('.ball').position().left + params.distance) + 'px'}, 200);

});

**ET VOILA**

Conclusion

* Beaucoup de configuration pour un premier projet mais ensuite → copier coller du template et ça va très vite.
* Énormément d’avantages pour grunt (amélioration de la vitesse du site, de la qualité du code...etc.)
* Existence de générateur de template de projet comme Yeoman → très intéressant car gain de temps
* Alliance NodeJS + Socket.io pour du temps réel c’est le must et le plus simple.