**Université Saad DAHLAB - Blida 1**

**Faculté des sciences**

**Département d’Informatique**

Master 1 : Sécurité des systèmes d'information

*Sujet :*

implementation Systemes de Communication.

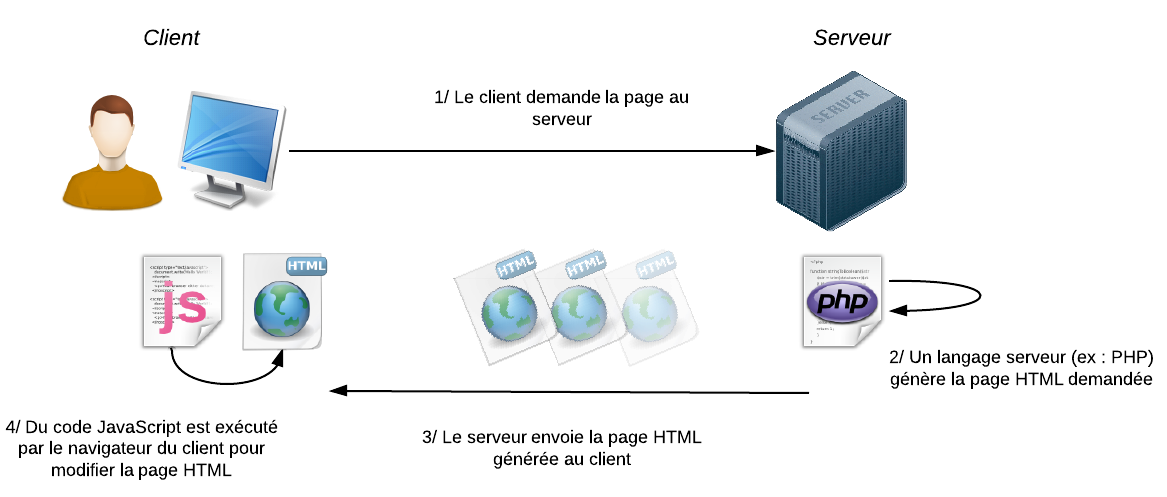
*Présenté par :* *Encadré par :*

MEHAL Anis Madame BOUSTIA.N

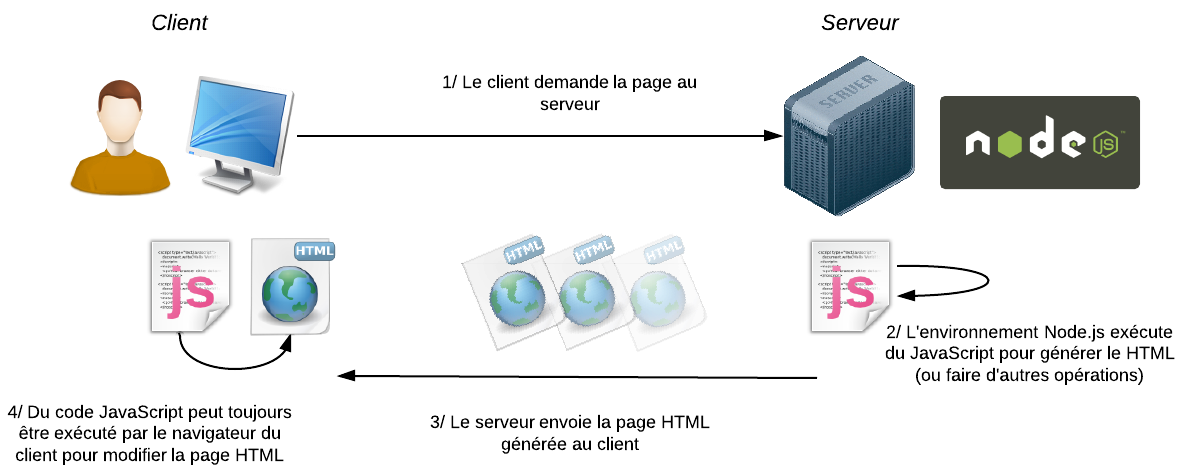
Année universitaire : 2019/2020

**Node js :**

Jusqu'ici, **JavaScript avait toujours été utilisé du côté du client**, c'est-à-dire du côté du visiteur qui navigue sur notre site. Le navigateur web du visiteur (Firefox, Chrome, IE...) exécute le code JavaScript et effectue des actions sur la page web.



Par contre, **Node.js offre un environnement côté serveur** qui nous permet aussi d'utiliser le langage JavaScript pour générer des pages web. En gros, il vient en remplacement de langages serveur comme PHP, Java EE, etc.

****

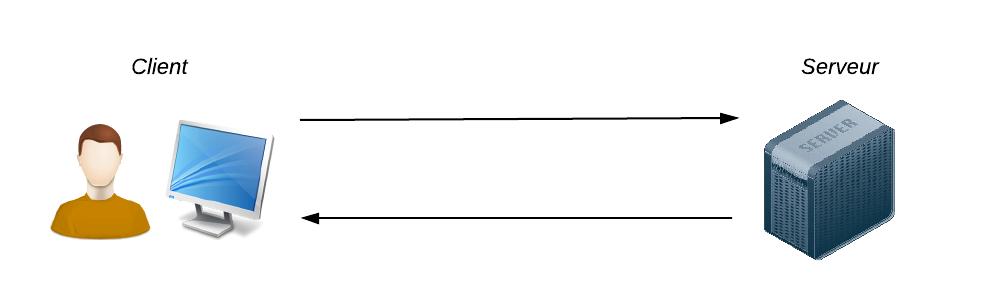
Avec Node.js, vous pouvez créer des applications rapides comme :

* Un serveur de Chat
* Un système d'upload très rapide
* ... et de façon générale n'importe quelle application qui doit répondre à de nombreuses requêtes rapidement et efficacement, en temps réel

**socket.io :**

C'est une bibliothèque qui nous simplifie beaucoup les choses, socket.io se base sur plusieurs techniques différentes qui permettent la communication en temps réel

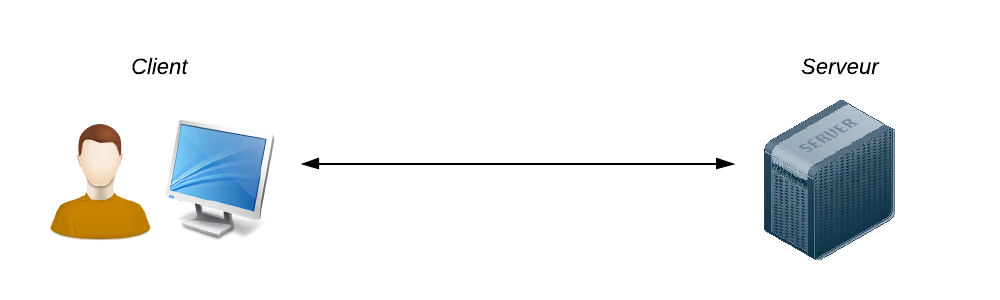
Habituellement, sur le Web, la communication est asynchrone. Le Web a toujours été conçu comme ça : le client demande et le serveur répond.

****

**WebSocket** est une fonctionnalité supportée par l'ensemble des navigateurs récents. Elle permet un **échange bilatéral synchrone** entre le client et le serveur.

**socket.io** nous permet d'utiliser les WebSockets très facilement.

**WebSocket** est une nouveauté du Web qui permet de laisser une sorte de "tuyau" de communication ouvert entre le client et le serveur. Le navigateur et le serveur restent connectés entre eux et peuvent s'échanger des messages dans un sens comme dans l'autre dans ce tuyau. Désormais, le serveur peut donc lui-même décider d'envoyer un message au client comme un grand !



**Emettre et recevoir des messages avec socket.io:**

Quand on utilise socket.io, on doit toujours s'occuper de deux fichiers en même temps :

* Le fichier serveur (ex : app.js) : c'est lui qui centralise et gère les connexions des différents clients connectés au site.
* Le fichier client (ex : index.html) : c'est lui qui se connecte au serveur et qui affiche les résultats dans le navigateur.

##### **Le serveur (app.js) :**



Ce code fait 2 choses :

* Il renvoie le fichier index.html quand un client demande à charger la page dans son navigateur.
* Il se prépare à recevoir des requêtes via socket.io. Ici, on s'attend à recevoir un seul type de message : la connexion. Lorsqu'on se connecte via socket.io, on logge ici l'information dans la console.

**Le client (index.html) :**

Intéressons-nous maintenant au client. Le fichier index.html est envoyé par le serveur node.js. C'est un fichier HTML tout ce qu'il y a de plus classique, si ce n'est qu'il contient un peu de code JavaScript qui permettra ensuite de communiquer avec le serveur en temps réel via socket.io :



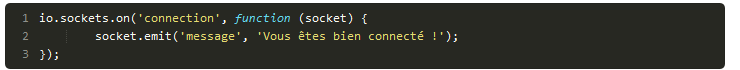
#### **Envoi et réception de messages :**

Maintenant que le client est connecté, on peut échanger des messages entre le client et le serveur. Il y a 2 cas de figure :

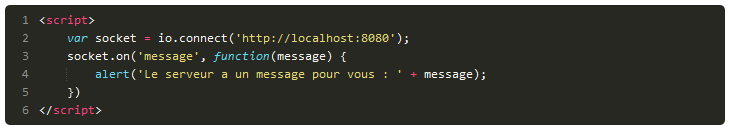
* Le serveur veut envoyer un message au client
* Le client veut envoyer un message au serveur

##### **Le serveur veut envoyer un message au client**

Je propose que le serveur envoie un message au client lorsqu'il vient de se connecter, pour lui confirmer que la connexion a bien fonctionné. Rajoutez ceci au fichier app.js :

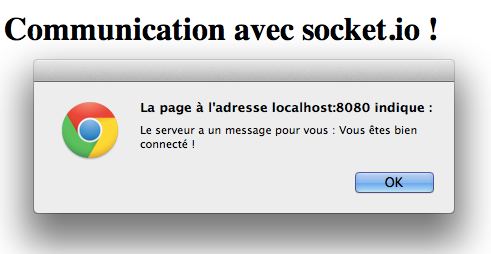


Du côté du fichier index.html (le client), on va écouter l'arrivée de messages de type "message" :



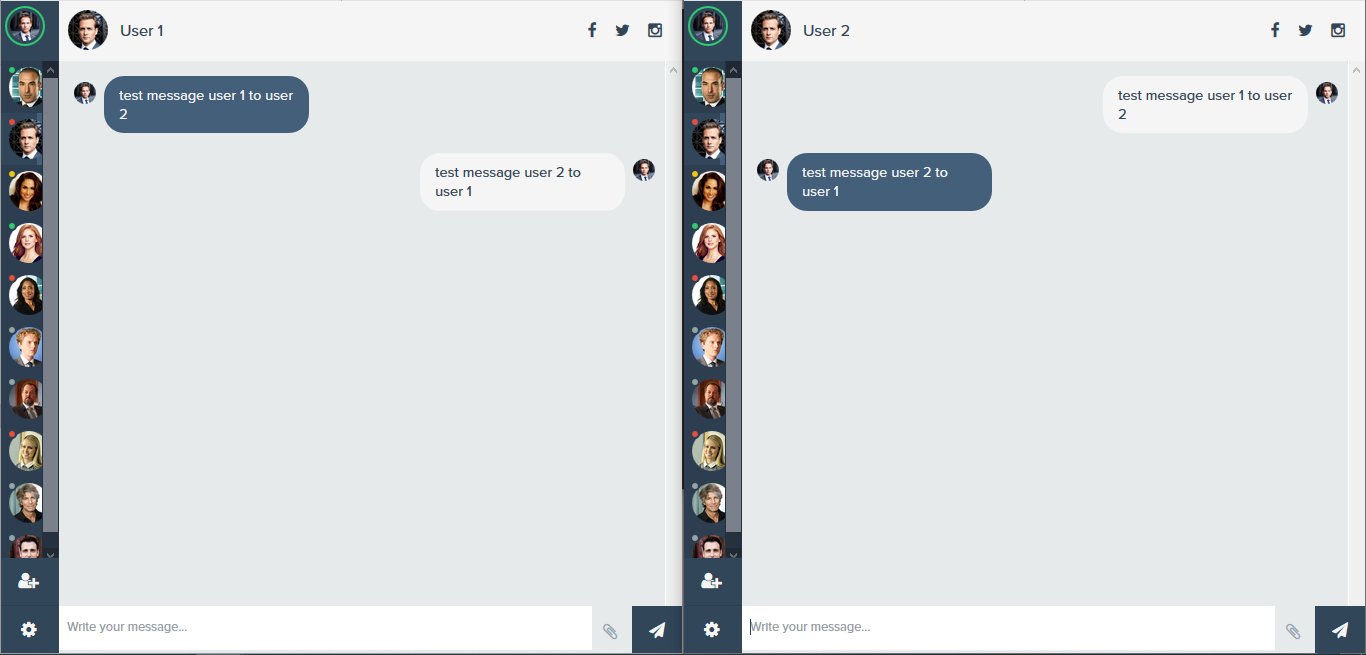
Avec socket.on(), on écoute les messages de type "message". Lorsque des messages arrivent, on appelle la fonction de callback qui, ici, affiche simplement une boîte de dialogue.

Essayez, vous verrez que lorsque vous chargez la page index.html, une boîte de dialogue s'affiche indiquant que la connexion a réussi !



**Exemple (ç'est notre projet) :**

Voilà exemple de discussion entre deux joueur

****