



S

1 –

2 –

2

2

2

# CRÉEZ VOTRE CHAT NODEJS AVEC SOCKET.IO

... 2

... 3

... 4

... 4

... 3

2.4 – Les boucles « for » ..... 3

*Créez votre propre chat instantané avec nodejs  
et socket.io*

4.2 – Serveur HTML ..... 8

5 – Faites votre bot inutile parfait à votre tour ! ..... 9



# socket.io

Coding Club Epitech Toulouse



# 1 – Qu'est-ce que Nodejs et Socket.io

Node.js est une plateforme de développement JavaScript. Socket.io est un outil javascript permettant d'envoyer des messages et de communiquer rapidement.

Ici nous allons nous servir de Node.js et Socket.io pour créer un système de chat instantané sur navigateur.



## 2 – Les particularités de la syntaxe avec Node.js

### 2.1 – Les méthodes pour traiter une chaîne de caractères

`.split()` permet de transformer une chaîne en tableau en séparant les mots de la chaîne par un séparateur. Par exemple :

« Je suis une chaîne » nous donne après avoir utilisé `split` par espace : [« Je », « suis », « une », « chaîne »].

`.toLowerCase()` permet de passer toutes les lettres capitales d'une chaîne en petites lettres :

« Je SUIS UNE chaîne » nous donnera alors « je suis une chaîne ».

### 2.2 – Les méthodes pour traiter les tableaux

Pour traiter les tableaux nous allons utiliser `.indexOf` dans ce tuto, cette méthode renvoie le premier indice pour lequel on trouve un élément donné dans un tableau.

### 2.3– Les boucles « for »

Les boucles « `for` » avec le mot clé « `of` » vont nous permettre de créer une boucle sur un tableau par exemple. Voici un exemple de boucle « `for` » :

```
for (word of words)
{ //faire des trucs }
```



Créez votre chat avec Node.js et Socket.io – Coding Club Epitech Toulouse

Ce qu'il va se passer c'est que la variable `word` va prendre la valeur suivante dans le tableau `words` à chaque tour de boucle, la boucle s'arrête quand il n'y a plus de valeurs dans le tableau.



## 3 – Installation des outils Node.js

---

**Si vous êtes sur linux**, ouvrez un terminal et exécutez les commandes suivantes :

```
sudo apt-get install npm  
sudo npm install -g n
```

Lancez ensuite un terminal :

```
s
```

Tapez ensuite ces commandes pour installer socket.io dans votre dossier favori:

```
npm install init  
npm install --save socket.io
```



## 4 – Commencez à coder !

---

VSCode

### 4.1 – Le framework web

Pour commencer nous allons créer une page web HTML simple qui propose un formulaire et une affiche une liste de messages. Pour cela nous allons utiliser un framework web Node.js : express. Pour rappel, HTML est le langage utilisé pour créer des pages Web. Si vous voulez en savoir plus sur l'HTML, jetez un œil au tuto HTML que vous nous avons fourni.

Commençons par créer un dossier « chat-exemple ». Nous allons ensuite créer un fichier « package.json » qui contient des informations sur notre projet :

```
{
  "name": "socket-chat-example",
  "version": "0.0.1",
  "description": "my first socket.io app",
  "dependencies": {}
}
```

Afin de récupérer et d'installer Express, l'outil qui va nous permettre d'afficher les pages web, lancez la commande :

```
npm install --save express@4.15.2
```

Maintenant que express est installé nous pouvons créer un fichier « index.js » qui va nous permettre de configurer notre application :

```
var app = require('express')();
var http = require('http').Server(app);

app.get('/', function(req, res){
  res.send('<h1>Hello world</h1>');
});

http.listen(3000, function(){
  console.log('listening on *:3000');
});
```

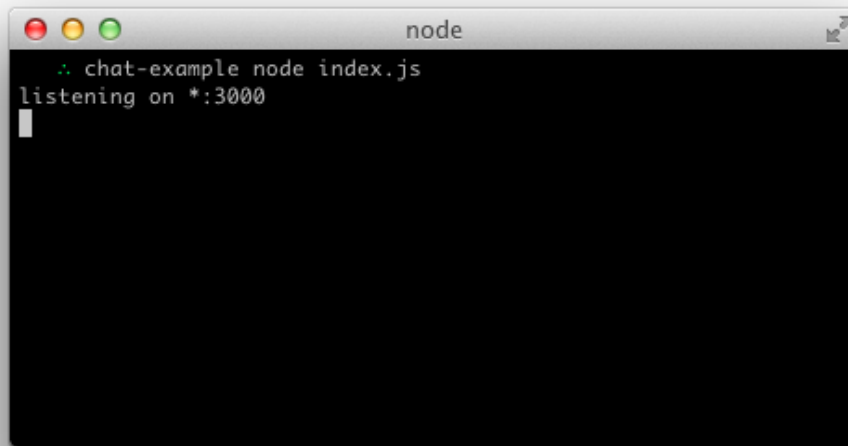
Ce code nous permet de faire 3 choses différentes :



## Créez votre chat avec Node.js et Socket.io – Coding Club Epitech Toulouse

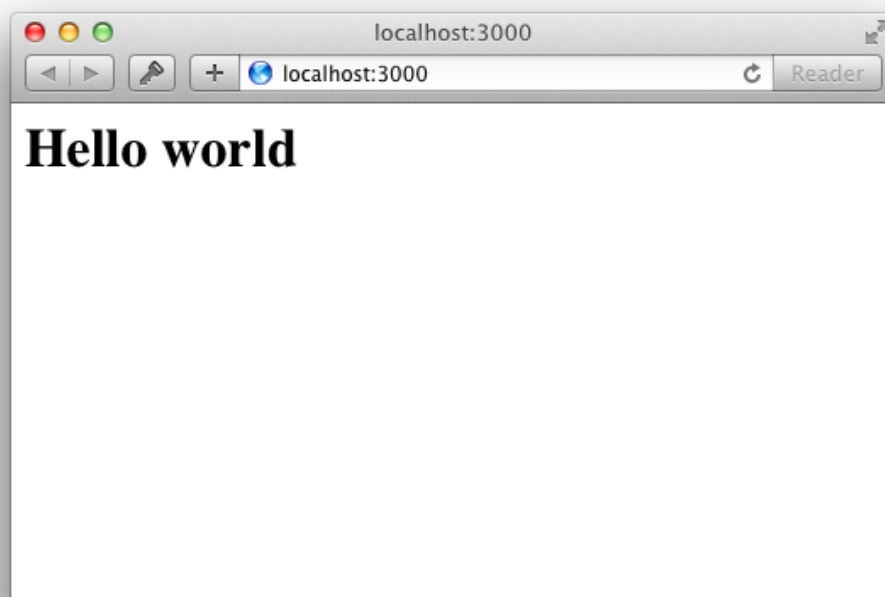
- Express initialise notre application pour qu'elle soit un gestionnaire de fonctions que nous pourrions fournir à un serveur HTTP (ligne 2).
- On définit une route par défaut « / » que l'on pourra appeler afin d'aller sur la page d'accueil de notre site (ligne 3).
- On fait écouter notre port HTTP sur le port 3000 (ligne 6).

Si vous lancez `node index.js` vous devriez voir quelque chose comme ça :



```
node
chat-example node index.js
listening on *:3000
```

Et si vous ouvrez cet url `http://localhost:3000` dans votre navigateur vous verrez quelque chose comme ça :





## 4.2 – Serveur HTML

Jusqu'à maintenant dans « index.js » on appelle « res.send » et on le passe à une chaîne de caractère HTML. Notre code ne serait pas très clair si on plaçait tout notre code HTML dans ce fichier. A la place nous allons créer un fichier « index.html ».

Nous allons donc changer nos lignes de codes permettant de créer une route par défaut afin d'utiliser `sendFile` à la place :

```
app.get('/', function(req, res){  
  res.sendFile(__dirname + '/index.html');  
});
```

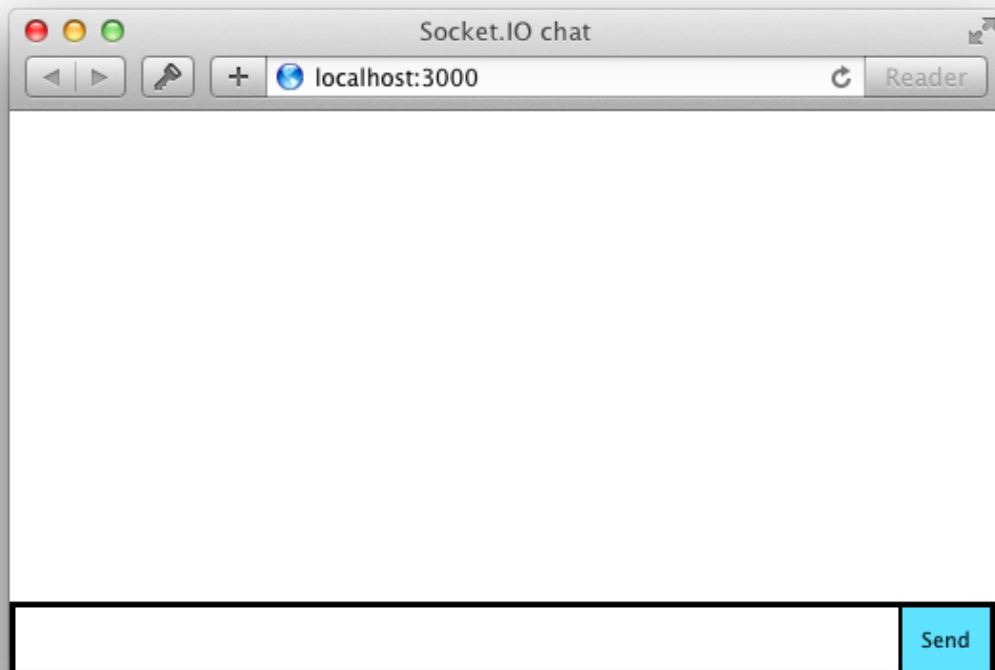
Et modifier notre fichier `index.html` avec le code suivant :

```
<!doctype html>  
<html>  
  <head>  
    <title>Socket.IO chat</title>  
    <style>  
      * { margin: 0; padding: 0; box-sizing: border-box; }  
      body { font: 13px Helvetica, Arial; }  
      form { background: #000; padding: 3px; position: fixed; bottom: 0; width: 100%; }  
      form input { border: 0; padding: 10px; width: 90%; margin-right: .5%; }  
      form button { width: 9%; background: rgb(130, 224, 255); border: none; padding: 10px; }  
      #messages { list-style-type: none; margin: 0; padding: 0; }  
      #messages li { padding: 5px 10px; }  
      #messages li:nth-child(odd) { background: #eee; }  
    </style>  
  </head>  
  <body>  
    <ul id="messages"></ul>  
    <form action="">  
      <input id="m" autocomplete="off" /><button>Send</button>  
    </form>  
  </body>  
</html>
```





Maintenant si on recharge la page (en appuyant sur Ctrl+C et en relançant `node index`), cela devrait donc ressembler à ça :



## 4.3 – Intégration de Socket.IO

Socket.IO est composé de deux parties :

- Un serveur qui s'intègre sur le serveur HTTP Node.JS : `socket.io`.
- Une bibliothèque client qui est utilisé du côté navigateur : `socket.io-client`.

Au cours du développement `socket.io` s'occupe de la connexion entre le client et le serveur pour nous. Pour le moment nous avons juste à installer un module :

```
npm install --save socket.io
```

Cela va installer le module et ajouter la dépendance au fichier `package.json`.



Nous allons maintenant changer `index.js` pour y ajouter ce code :

```
var app = require('express')();
var http = require('http').Server(app);
var io = require('socket.io')(http);

app.get('/', function(req, res){
  res.sendFile(__dirname + '/index.html');
});

io.on('connection', function(socket){
  console.log('a user connected');
});

http.listen(3000, function(){
  console.log('listening on *:3000');
});
```

Ici on initialise une instance de `socket.io` en y passant un objet `http`. Le serveur écoute et attend un évènement appelé `connexion`, une fois qu'il a reçu une connexion il l'envoie dans la console.

Maintenant nous allons ajouter le code suivant à notre fichier `index.html` avant la balise `</body>` :

```
<script src="/socket.io/socket.io.js"></script>
<script>
  var socket = io();
</script>
```

C'est tout ce dont nous avons besoin pour charger le `socket.io-client`, ce qui va nous connecter au réseau socket.

On remarque aussi que l'on ne spécifie aucune URL à l'appel de `io()` car il appelle par défaut l'URL de notre page d'accueil.

Maintenant si on actualise le serveur et la page d'accueil (`http://localhost:3000`) on devrait voir la console afficher « user connected ». Si on essaye d'ouvrir plusieurs terminaux, plusieurs messages seront affichés dans la console.



Chaque socket utilise aussi un évènement spécial `disconnect`.

```
io.on('connection', function(socket){
  console.log('a user connected');
  socket.on('disconnect', function(){
    console.log('user disconnected');
  });
});
```

Et si on rafraîchit des terminaux on verra ceci dans la console :

```
node
chat-example node index.js
listening on *:3000
a user connected
user disconnected
a user connected
user disconnected
a user connected
```

## 4.4 – Emettre des évènements

L'idée principale de socket.IO est que l'on peut envoyer et recevoir tous les évènements que l'on veut, avec n'importe quelles données. N'importe quel objet qui peut être encodé en json peut être utilisé, même un fichier binaire peut-être supporté.

Nous allons faire en sorte que lorsque l'utilisateur tape un message, le serveur le reçoit en tant qu'un évènement `chat message`. La section de script dans le fichier `index.html` doit comporter ce code :

```
<script src="/socket.io/socket.io.js"></script>
<script src="https://code.jquery.com/jquery-1.11.1.js"></script>
<script>
  $(function () {
    var socket = io();
    $('form').submit(function(){
      socket.emit('chat message', $('#m').val());
      $('#m').val('');
      return false;
    });
  });
</script>
```



Et dans notre fichier index.js on utilise ce code afin d'afficher l'évènement `chat message` :

```
io.on('connection', function(socket){
  socket.on('chat message', function(msg){
    console.log('message: ' + msg);
  });
});
```

Maintenant vous devriez voir dans la console ce que vous tapez sur votre page d'accueil.

## 4.5 – Diffusion

Le but suivant de notre tutoriel est d'émettre un évènement depuis notre serveur pour le reste des utilisateurs.

Afin d'envoyer un évènement à tout le monde, socket.IO nous fournit `io.emit` :

```
io.emit('some event', { for: 'everyone' });
```

Si nous voulons envoyer un message à tout le monde excepté pour un certain socket, nous avons le flag `broadcast` :

```
io.on('connection', function(socket){
  socket.broadcast.emit('hi');
});
```

Dans notre cas, nous allons envoyer un message à tout le monde, même l'émetteur :

```
io.on('connection', function(socket){
  socket.on('chat message', function(msg){
    io.emit('chat message', msg);
  });
});
```



Du côté client lorsque l'on reçoit un évènement `chat message` nous allons l'inclure dans la page. Le code JavaScript du côté client devrait maintenant ressembler à ça :

```
<script>
$(function () {
  var socket = io();
  $('form').submit(function(){
    socket.emit('chat message', $('#m').val());
    $('#m').val('');
    return false;
  });
  socket.on('chat message', function(msg){
    $('#messages').append($('- ').text(msg));
  });
});
</script>

```

Et notre application de chat est terminée !!

## 5 – Améliorer notre chat

- Envoyer un message aux utilisateurs connectés lorsque quelqu'un se connecte ou se déconnecte.
- Ajouter des surnoms aux utilisateurs.
- Ne pas envoyer le message à l'émetteur de celui-ci, seulement l'afficher (comme dans un vrai chat).
- Ajouter une fonctionnalité « {user} est en train d'écrire ».
- Montrer qui est en ligne.
- Ajouter un système de messages privés.
- Et tout ce que vous voulez d'autre pour faire un super chat !!



**Écriture tutoriel :** Guillaume R, Tifaine L, Bastien H

