

Table des matières

[Introduction 2](#_Toc8631947)

[Qui suis-je ? 2](#_Toc8631948)

[Le contexte 3](#_Toc8631949)

[L’influence 4](#_Toc8631950)

[Le concept 5](#_Toc8631951)

[Compétences à couvrir 6](#_Toc8631952)

[Front-end 6](#_Toc8631953)

[Back-end 6](#_Toc8631954)

[Cahier des charges 7](#_Toc8631955)

[Cas pratique 8](#_Toc8631956)

[English 8](#_Toc8631957)

[Introduction 8](#_Toc8631958)

[The web framework 8](#_Toc8631959)

[Serving HTML 10](#_Toc8631960)

[Integrating Socket.IO 11](#_Toc8631961)

[Emitting events 12](#_Toc8631962)

[Broadcasting 13](#_Toc8631963)

[Français 13](#_Toc8631964)

[Introduction 14](#_Toc8631965)

[Le framework web 14](#_Toc8631966)

[Servir du HTML 15](#_Toc8631967)

[Intégrer Socket.IO 16](#_Toc8631968)

[Emettre des évènements 18](#_Toc8631969)

[Distribuer 18](#_Toc8631970)

[Aller plus loin 19](#_Toc8631971)

# Introduction

## Qui suis-je ?

Je m’appelle Hubert Pfersdorff, j’ai 26 ans. Actuellement en formation au titre RNCP niveau 3 développeur web et web mobile au sein de l’organisme de formation Elan à Strasbourg, j’ai réalisé cette plateforme pour mettre à l’épreuve mes compétences.

Fasciné depuis l’adolescence par la science-fiction, la technologie mais aussi les classiques notamment au cinéma, j’ai naturellement été attiré vers les métiers du web.

J’ai aujourd’hui l’opportunité de conduire un projet du concept à la présentation, m’autorisant une liberté totale sur les moyens et la réalisation. Cette liberté m’a mené à rapidement instaurer une rigueur et un cadre de travail spécifique afin de ne pas perdre mon fil rouge.

Avant de démarrer la formation, je n’avais pas de notions en développement orienté objet. Mener ce projet à bien m’a permis de découvrir énormément de choses : j’ai conçu chaque aspect comme si j’étais une petite équipe dans une agence. J’ai pris du temps et du plaisir pour chaque chose, de la matérialisation de mes idées à la production. Je suis comme ça. J’aime les petits détails, que ce soit sur un outil ou une personne. Le cadre de travail est d’autant plus important !

En abordant un projet, j’avais tendance à m’engouffrer dans le développement de détails. Exemple : un carnet de transmission pour professionnels de la santé, mon meilleur code se situe dans l’enregistrement de chaque action ou exception dans un fichier de logs, consultable dans une zone admin.

Je n’ai pas changé, mais aujourd’hui afin de me préparer au mieux pour mon futur travail, j’aborde chaque chose de manière professionnelle. Ce raisonnement a je crois porté ses fruits : dans le cadre de ce projet, beaucoup a été réalisé en amont. Identité visuelle, concept, technologies à utiliser, déploiement, livraison, dossier de synthèse par exemple. C’est un nouvel exercice pour moi que je pense nécessaire peu importe la direction à prendre après la formation.

Cette plateforme est influencée par un certain nombre de choses sur lesquelles nous reviendrons. L’une des influences principales est mon expérience de 3 ans dans le milieu de l’hébergement web, du support technique mais aussi de la négociation. J’ai pensé chaque interface avec beaucoup d’empathie, car j’ai appris à me mettre à la place d’une autre personne. Aujourd’hui j’applique cette capacité à l’ergonomie de mes interfaces.

Je vous souhaite bonne lecture, et espère que les éléments que je vais vous apporter vous aideront à comprendre les choix faits ainsi que le produit final. (Une condition de lecture recommandée est l’écoute de *Comfortably Numb* lors du [live de Pink Floyd](https://www.youtube.com/watch?v=vi7cuAjArRs&list=RDvi7cuAjArRs&start_radio=1) à Londres en 1994, en 44Khz)

## Le contexte



Afin de m’approprier le projet et la volonté d’y investir beaucoup de temps, je dois être inspiré. J’ai pris du recul et me suis demandé ce que je recherchais quand arrive le moment d’une découverte. J’ai vais prendre l’exemple du film déjà vu ou non : aujourd’hui, je sais que l’émotion est le résultat que je recherche. Vais-je rire, pleurer, m’interroger ?

Moi qui ai maintenant l’opportunité de créer, que vais-je transmettre à l’utilisateur ? Ou plutôt, que vais-je proposer comme outil, afin de permettre aux utilisateurs de rechercher à leur tour une émotion. Je vais reprendre cette photo tirée du film *Dikkenek* d’Olivier Van Hoofstadt : pas besoin de chercher l’originalité, il faut simplement que ça me plaise. L’idée m’est venue par hasard et j’en suis fier. Attention, je ne prétends pas avoir trouvé l’idée qui révolutionnera notre Internet. C’est une idée qui simplement me plait, mais surtout qui m’inspire. J’y viens.

Nous connaissons tous Facebook, Twitter, Instagram, Reddit qui nous permettent de tout partager. J’ai pensé que me baser sur un réseau social pour mon projet de fin de formation serait une base solide, me donnant assez de matière à défendre. Mais il manque le grain de sel. Et c’est ça l’idée, le grain de sel. De nos vies, mais en particulier le grain de sel de nos interactions avec notre environnement.

Une plateforme de partage de sel. De l’anecdote originale aux scènes les plus farfelues, l’outil permet de faire profiter les autres d’un moment éphémère. Et comme j’aime les jeux, rendons tout ça interactif en permettant aux utilisateurs de jouer avec l’histoire.

On m’interpellera en me demandant « mais pourquoi aller sur **Strapp**, alors que j’ai déjà Facebook ou twitter avec mes amis ? ». Pourquoi utilise-t-on un four pour faire le gratin, alors qu’un incendie nous permet l’accès au même résultat ? Le four est adapté, et tous les utilisateurs de four tendent vers le même but : manger.

Sur **Strapp** c’est pareil : je suis une plateforme dédiée à l’original, à l’éphémère, je recueille les histoires et constitue une véritable carte de l’imprévu. Demain vous visitez une nouvelle ville, pimentez votre séjour en vous basant sur mes données et mes utilisateurs !

## C:\Users\Stagiaire\Pictures\18869843.jpg-r_1280_720-f_jpg-q_x-xxyxx.jpgL’influence

Le caractère visuel est un aspect très important pour moi, j’ai besoin de me sentir bien dans mon projet pour me l’approprier, et ce dès le début. J’ai besoin d’avoir en tête un résultat afin d’être efficace.

Après plusieurs essais, je me suis arrêté sur le raisonnement suivant : évènements dans la rue + éphémère + interaction + étrange = Les rues de New York + film éphémère + communication, échange + ville la nuit = *Taxi Driver* de Martin Scorsese. Les couleurs des taxis new-yorkais me plaisent, le film me plait, l’ambiance qu’il dégage me plait.

Sans rentrer dans les détails du film, je m’imagine au volant de son taxi assister à une scène aussi loufoque qu’indescriptible. Pourquoi ne pas baser la plateforme sur cette ambiance ? Aussi, n’étant pas graphiste, je ne prétends pas pouvoir créer une identité visuelle de toute pièce, « from scratch », m’inspirer d’une œuvre que j’aime m’aide énormément.

## Le concept

Le concept est simple : un réseau social où partager un moment original sous la forme d’un vote, que nous appellerons « story ». Les autres utilisateurs ont la possibilité de voter s’ils pensent que l’histoire est vraie ou fausse. L’issue du vote détermine le degré de fiabilité de l’auteur, ou sa capacité à raconter une fantaisie. L’issue du vote détermine aussi la capacité des autres à déterminer la véracité d’une histoire. Ainsi, chaque utilisateur dresse son profil en fonction de ses actions.

Le partage de l’histoire se fait sous la forme d’un billet comprenant un titre, la description, ainsi qu’une photo de la scène. L’auteur a la possibilité de choisir une durée, pendant laquelle le vote sera visible de tous sans la photo. Cette photo n’est visible par les votants qu’une fois le vote achevé, ainsi que l’issue du vote et la vérité. Le billet de la story s’affiche sous la forme d’un marqueur sur la carte.

Ces données serviront à dresser des zones géographiques sur la carte affichant les lieux denses en scènes originales, ou en grand raconteurs d’histoires. Je doute que le deuxième filtre soit très pertinent, mais il peut être intéressant de voir se révéler des zones qui inspirent l’imagination. Voilà qui couvre la partie carte et story.

Coté réseau social, une fois inscrit, chaque utilisateur a la possibilité de voir le profil d’autres utilisateurs et de les ajouter à sa liste d’amis. Une fois amis, deux utilisateurs amis peuvent interagir par le biais des story, des commentaires et plus tard de la messagerie.

Une page de fil d’actualité est l’affichage pour l’utilisateur où il peut voir les stories récemment postées par ses amis où les personnes qu’ils suit, ainsi que les informations basiques telles que le nombre de commentaires, le nombre de votes, et le temps restant.

Au clic sur une story nous accédons aux détails de celle-ci, permettant une vie plus statique des différentes informations.

Le cœur de l’application est la carte. Même sans être un utilisateur enregistré, la page d’accueil affiche déjà une carte, avec par exemple les stories populaires accessibles à tous, vue uniquement.

Une fois enregistré, tout tourne autour de la carte. On y affiche les stories proches de soit, avec par exemple un filtre sur la ville. Chaque point sur la carte est cliquable pour accéder aux informations basiques de la story, avec un lien menant vers la page des détails.

Quand un utilisateur créé une story, il doit choisir une durée qui générera une date de fin. Une fois la date dépassée, l’interaction avec la story n’est plus possible, et les votes sont révélés, ainsi que la véracité de celle-ci, générant un score pour l’auteur et les votants en fonction des résultats.

# Compétences à couvrir

Le projet étant réalisé seul, il permet d’étendre au maximum la couverture des compétences à couvrir pour le diplôme. Voici les compétences abordées dans mon projet :

## Front-end

* Maquetter une application.
  + **Strapp** a été maquetté à l’aide de d’Adobe XD et Adobe Photoshop. Ayant quelques notions sur Photoshop, je n’ai pas eu grand mal à prendre en main XD. Le logiciel m’a permis de dresser un visuel de mes pages en amont, avec une gestion des différents chemins et transitions.
* Réaliser une interface utilisateur web statique et adaptable.
  + Pour gérer mon front, j’étais partis pour m’aider du framework CSS Semantic.UI, mais je me suis rabattu sur Bootstrap, plus adapté à la flexibilité dont j’aurai besoin. L’application étant mobile first, j’ai dû gérer la totalité du responsive moi-même.
* Développer une interface utilisateur web dynamique.
  + L’application étant principalement composée de formulaire, sous la forme de simples boutons jusqu’aux pages entières, l’utilisation de Javascript et d’Ajax m’a aidé à rendre le parcours de l’utilisateurs plus fluide et agréable. Aussi, mon système de notifications reflète bien l’aspect dynamique de l’application, permettant la mise à jour de données coté client sans action requise de sa part.

## Back-end

* Créer une base de données.
  + Le projet étant réalisé avec Symphony 4, tout tourne autour de la base de données et des relations entre les entités qui y sont stockées. L’appli est basée sur une base de données MySQL 5.7, propulsée par le moteur InnoDB.
* Développer les composants d’accès aux données.
  + Symphony ici nous aide en apportant l’implémentation très profonde de l’ORM Doctrine, me permettant de manipuler directement des entités générées depuis la base de données. Doctrine permet aussi faire beaucoup à la main, par exemple lorsque je n’ai pas besoin d’une entité entière.
* Développer la partie back-end d’une application web ou web mobile.
  + L’application entière est back-end, puisque son utilisation requiert un compte. Une interface d’administration vient compléter cet aspect.

# Cahier des charges

## Objectifs du site

L’objectif premier de **Strapp** est d’offrir aux utilisateurs un lieu d’échange d’anecdotes vraies ou fausses ayant pris part de préférence en publique. Ce lieu d’échange permet aux utilisateurs d’interagir par le biais des « stories », messages, commentaires et votes.

## Périmètre

L’application est conçue pour une utilisation mobile avant tout, permettant aux utilisateurs de s’y retrouver facilement sur leurs appareils dans la rue, avec une interface accessible entièrement d’une main.

L’application est pour le moment en anglais, afin de permettre à un maximum l’accès au contenu. L’interface est en anglais, le contenu sera dans la langue des utilisateurs. Je suis très curieux de voir où il sera utilisé une fois en production !

## Objectifs quantitatifs

Le trafic attendu n’est pas encore connu, étant donné qu’une plateforme semblable n’existe pas encore.

## Cible

Le publique visé a été pensé comme suit : j’ai la chance d’avoir dans ma famille des proches curieux. Curieux sur leurs origines, sur l’histoire de notre région et de nos ancêtres. Je n’ai pas encore développé cette même curiosité, mais leur façon de parler de petites anecdotes passées m’a inspiré. Le public visé est donc le curieux, celui qui s’intéresse aux détails de ce qui l’entoure. C’est un publique qui a entre 20 et 70 ans !

## Existant

Le nom de domaine choisit est strapp.xyz, enregistré chez Online. L’application sera hébergée sur un VPS dans la phase de tests puis serveur dédié chez OVH.

## Charte graphique

Les codes couleur ont évolué plusieurs fois avant que je m’arrête sur une combinaison satisfaisante :

#ffda00 (RGB Décimal 255, 218, 0) (Pure (or mostly pure) yellow),

#121212 (RGB Décimal 18, 18, 18) (Very dark gray (mostly black)),

#ffffff (RGB Décimal 255, 255, 255) (white)

Polices utilisées : Porto, Roboto, Fairplay Display.

## Design

« Good design is as little design as possible » - Dieter Rams

L’objet principal de l’application étant le contenu partagé par les utilisateurs, son design a pour objectif de faciliter l’accès à celui-ci. La majeure partie de l’utilisation de l’application ayant comme media des formulaires et des boutons, ceux-ci doivent avoir du retour et être satisfaisant à l’utilisation. Chaque action utilisateur validée est accompagnée d’une notification, qui vient compléter les animations visuelles.

## Logo

L’objectif de la charte graphique est de rappeler les taxi New-yorkais, à l’aide du jaune et du noir.



A gauche le logo principal, à droite le favicon utilisé pour l’application. Logo basique mais que remplit son rôle, être reconnaissable.

# Cas pratique

## English

In order to implement a way for users to communicate, I had to look for a new technology allowing me to handle real time requests. I have already worked with AJAX (PHP and jQuery) but it seemed too heavy to operate a chat server. I knew I had to use Node.js, which is the perfect platform to do such things, but it does not allow real time queries by itself (update one client after another triggers a listener).

Socket.IO is my solution (see <https://socket.io/>). Here I have for you the documentation I followed in order to setup my first chat that would later become my chatroom server used in Strapp.xyz:

In this guide we’ll create a basic chat application. It requires almost no basic prior knowledge of Node.JS or Socket.IO, so it’s ideal for users of all knowledge levels.

### Introduction

Writing a chat application with popular web applications stacks like LAMP (PHP) has traditionally been very hard. It involves polling the server for changes, keeping track of timestamps, and it’s a lot slower than it should be.

Sockets have traditionally been the solution around which most real-time chat systems are architected, providing a bi-directional communication channel between a client and a server.

This means that the server can push messages to clients. Whenever you write a chat message, the idea is that the server will get it and push it to all other connected clients.

### The web framework

The first goal is to setup a simple HTML webpage that serves out a form and a list of messages. We’re going to use the Node.JS web framework express to this end. Make sure Node.JS is installed.

First let’s create a package.json manifest file that describes our project. I recommend you place it in a dedicated empty directory (I’ll call mine chat-example).

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

Now, in order to easily populate the dependencies with the things we need, we’ll use npm install --save:

npm install --save express@4.15.2

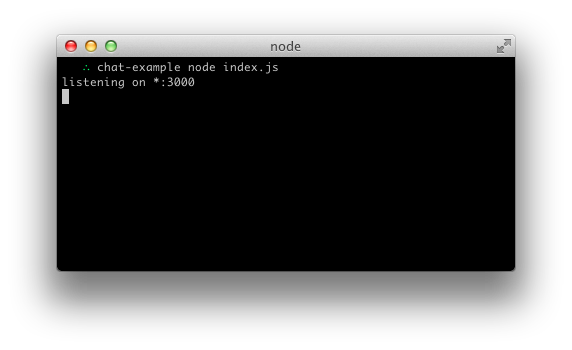
Now that express is installed we can create an index.js file that will setup our application.

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');  
});

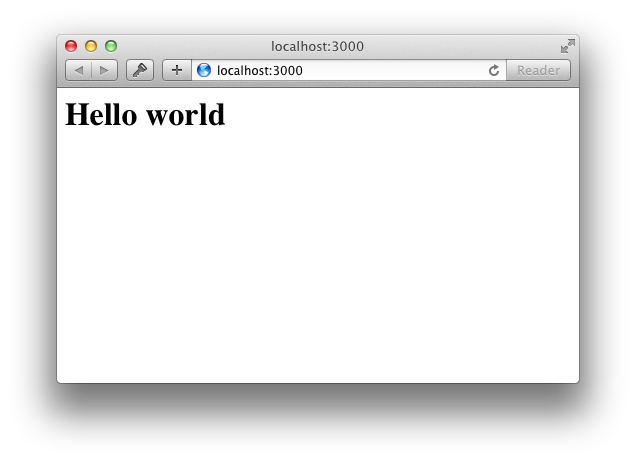
This translates into the following:

* Express initializes app to be a function handler that you can supply to an HTTP server (as seen in line 2).
* We define a route handler / that gets called when we hit our website home.
* We make the http server listen on port 3000.

If you run node index.js you should see the following:



And if you point your browser to <http://localhost:3000>:



### Serving HTML

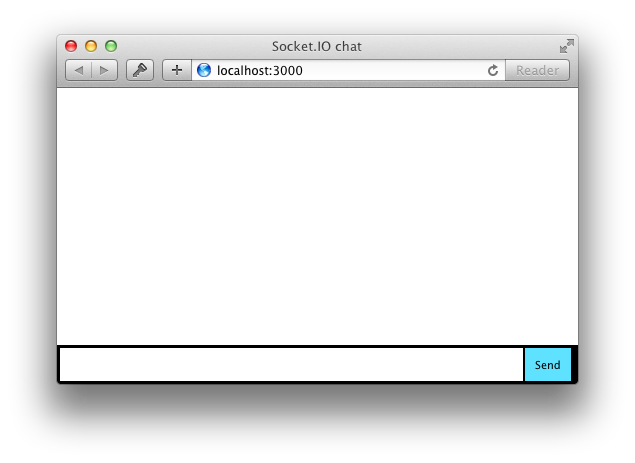
So far in index.js we’re calling res.send and pass it a HTML string. Our code would look very confusing if we just placed our entire application’s HTML there. Instead, we’re going to create a index.html file and serve it.

Let’s refactor our route handler to use sendFile instead:

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

And populate index.html with the following

If you restart the process (by hitting Control+C and running node index again) and refresh the page it should look like this:



<!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>

### Integrating Socket.IO

Socket.IO is composed of two parts:

* A server that integrates with (or mounts on) the Node.JS HTTP Server: socket.io
* A client library that loads on the browser side: socket.io-client

During development, socket.io serves the client automatically for us, as we’ll see, so for now we only have to install one module:

npm install --save socket.io

That will install the module and add the dependency to package.json. Now let’s edit index.js to add it:

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');});

Notice that I initialize a new instance of socket.io by passing the http (the HTTP server) object. Then I listen on the connection event for incoming sockets, and I log it to the console.

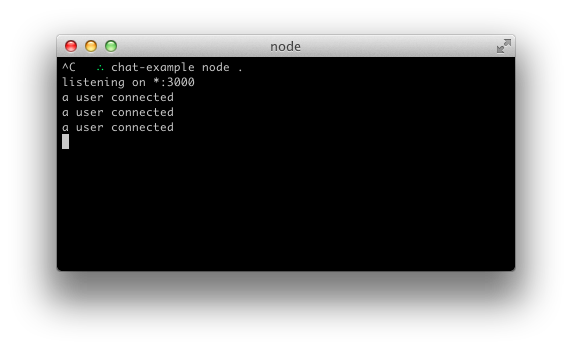
Now in index.html I add the following snippet before the </body>:

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

That’s all it takes to load the socket.io-client, which exposes a io global, and then connect.

Notice that I’m not specifying any URL when I call io(), since it defaults to trying to connect to the host that serves the page.

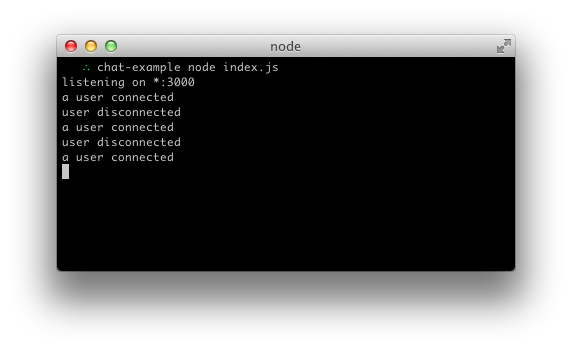
If you now reload the server and the website you should see the console print “a user connected”.

Try opening several tabs, and you’ll see several messages:

Each socket also fires a special disconnect event:

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

Then if you refresh a tab several times you can see it in action:



### Emitting events

The main idea behind Socket.IO is that you can send and receive any events you want, with any data you want. Any objects that can be encoded as JSON will do, and binary data is supported too.

Let’s make it so that when the user types in a message, the server gets it as a chat message event. The scripts section in index.html should now look as follows:

<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(e){  
 e.preventDefault(); // prevents page reloading  
 socket.emit('chat message', $('#m').val());  
 $('#m').val('');  
 return false;  
 });  
 });  
</script>

And in index.js we print out the chat message event:

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

### Broadcasting

The next goal is for us to emit the event from the server to the rest of the users.

In order to send an event to everyone, Socket.IO gives us the io.emit:

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

If you want to send a message to everyone except for a certain socket, we have the broadcast flag:

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

In this case, for the sake of simplicity we’ll send the message to everyone, including the sender.

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

And on the client side when we capture a chat message event we’ll include it in the page. The total client-side JavaScript code now amounts to:

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

And that completes our chat application, in about 20 lines of code!

## Français

Dans ce guide nous allons créer une application de chat basique. Ce guide n’ayant presque aucun prérequis sur Node.JS ou Socket.IO, il est ideal pour les utilisateurs de tous niveaux.

### Introduction

Il est admis que développer une application de chat avec des technologies comme L.A.M.P. peut s’avérer fastidieux. Ce qui inclus des modifications coté serveur ou encore prendre en compte les horodatages, il en résulte des temps d’exécution trop longs.

Les sockets ont traditionnellement été le cœur des solutions de chat en temps réel, fournissant un canal de discussion bidirectionnel entre le client et le serveur. Cela signifie que le serveur peut envoyer des messages au client. Dès que vous écrivez un message, le serveur le récupère et le distribue à tous les autres clients connectés.

### Le framework web

Le premier objectif est de mettre en place une simple page web HTML qui délivre un formulaire et une liste de messages.

Nous allons utiliser le framework de Node.JS Express pour y parvenir. Faites attention à ce que Node.JS soit installé.

Créons d’abord un fichier manifeste package.json qui décriera notre projet. Je vous recommande de la placer dans un dossier vide (je vais appeler le miens chat-example).

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

Pour maintenant peupler les dépendances avec ce dont nous avons besoin, nous allons utiliser npm install - -save.

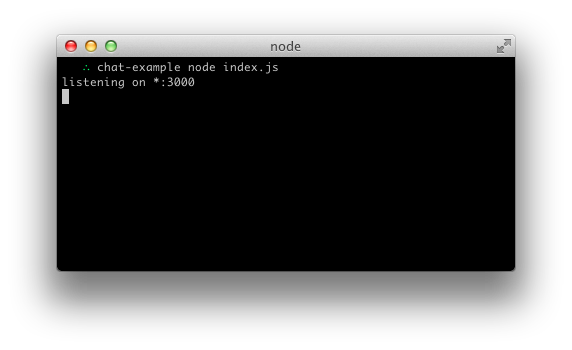
npm install --save express@4.15.2

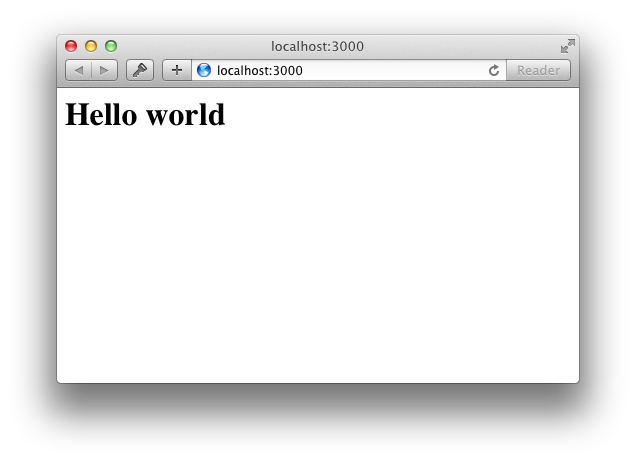
Maintenant qu’Express est installé nous pouvons créer un fichier index.js qui se chargera de notre application.

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 qui se traduit de la manière suivante :

* Express initialise app pour être notre gestionnaire de fonctions, soumettable à un serveur HTTP.
* Nous définissons notre gestionnaire de route / qui sera appelé à l’accueil du site.

En exécutant node index.js nous devrions avoir le résultat suivant : 

Dans notre navigateur sur http://localhost:3000: 

### Servir du HTML

Jusqu’ici dans notre index.js nous appelons res.send en lui fournissant une chaine de caractères HTML. Notre code serait illisible si nous y mettions tout notre HTML. Nous allons plutôt lui servir un fichier index.html.

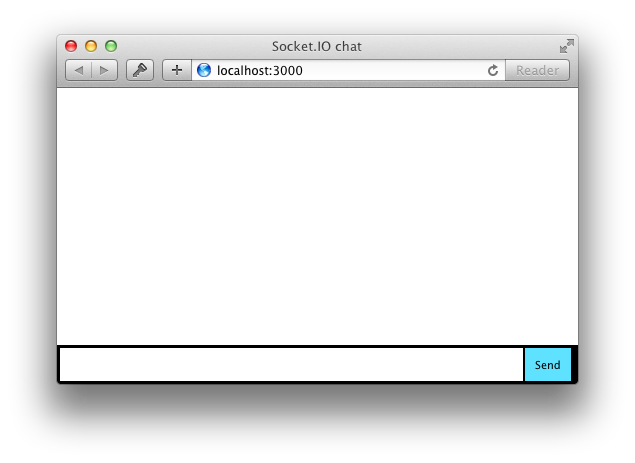
Corrigeons notre gestionnaire de route :

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

Notre index.html ressemble à ce qui suit :

<!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>

En redémarrant le serveur, voici le résultat :



### Intégrer Socket.IO

Socket.IO est composé de :

* Une surcouche du serveur http de Node.JS : socket.io
* Une librairie client chargée coté navigateur : socket.io-client

Pendant le développement, socket.io sert automatiquement le client pour nous, comme nous allons le voir, donc pour le moment nous n’avons besoin que d’un seul module :

npm install --save socket.io

Cette commande va installer le module et ajouter la dépendance dans package.json.

Editons maintenant index.js :

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');  
});

Notez que j’initialise une nouvelle instance de socket.io en lui passant l’objet http (serveur http). J’écoute ensuite l’évènement connexion pour des sockets entrants, que je log en console.

J’ajoute maintenant dans mon index.html avant </body> :

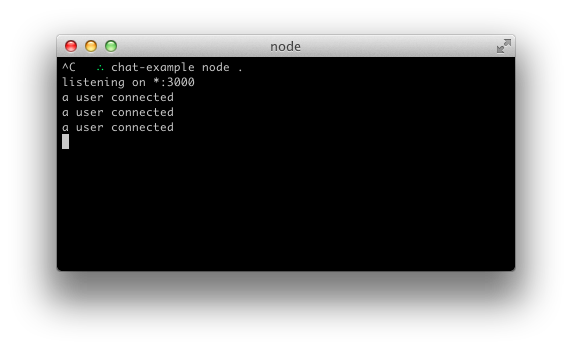
<script src="/socket.io/socket.io.js"></script>  
<script>  
 var socket = io();  
</script>

Voici tout ce qui est nécessaire coté client pour charger socket.io-client, ce qui expose la globale io et connecter.

Notez que je ne spécifie pas d’url quand j’appelle io(), qui par défaut tentera de se connecter à l’hôte qui sert la page.

Si vous redémarrez le serveur et le site vous devriez voir en console « a user connected ».

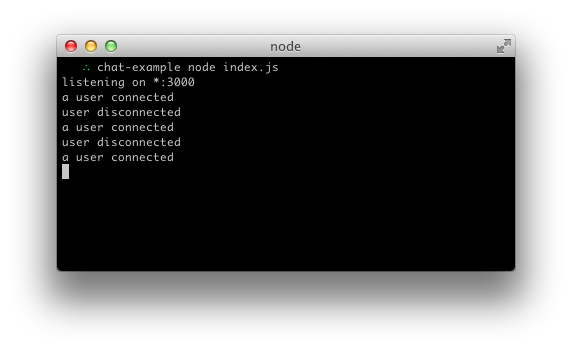
Essayez sur plusieurs onglets et vous verrez plusieurs messages :



Chaque socket lance aussi un évènement disconnect :

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

Si vous rafraichissez plusieurs fois la page vous pouvez le voir en action :



### Emettre des évènements

L’idée principale derrière Socket.IO est de pouvoir envoyer et recevoir n’importe quel évènement, avec n’importe quelle donnée. N’importe quel objet encodable en JSON ou donnée binaire sera supporté.

Faisons en sorte que lorsque l’utilisateur tape un message, le serveur l’interprète comme un évènement chat message. La section script dans index.html devrait ressembler à ceci :

<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(e){  
 e.preventDefault(); // prevents page reloading  
 socket.emit('chat message', $('#m').val());  
 $('#m').val('');  
 return false;  
 });  
 });  
</script>

Nous affichons ensuite l’évènement chat message dans index.js.

### Distribuer

L’objectif suivant pour nous est d’émettre l’évènement depuis le serveur aux autres utilisateurs. Afin d’envoyer un évènement à tout le monde, Socket.IO nous fournit io.emit :

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

Si vous souhaitez envoyer un message à tout le monde excepté un socket en particulier, nous avons le filtre broadcast :

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

Dans notre cas, par soucis de simplicité, nous allons envoyer le message à tout le monde, y compris l’expéditeur.

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

Coté client, il faut ensuite capturer l’évènement chat message et l’intégrer à la page. Le JavaScript final coté client est le suivant :

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

Voilà qui complète notre application de chat, en environ 20 lignes de code !

### Aller plus loin

* Avertir les utilisateurs lors de la connexion d’un nouvel utilisateur
* Gérer les pseudos
* Ajouter une fonctionnalité « un utilisateur est en train d’écrire »
* Montrer la liste des utilisateurs connectés
* Messagerie privée