

Codelab: Ajout de la fonctionnalité de chat avec Socket.IO

Objectifs

Dans ce codelab, vous explorez comment utiliser Node.js pour créer une application de chat en temps réel via une communication basée sur les événements via des sockets Web. Vous verrez comment les applications de chat peuvent être construites avec les outils HTML les plus simples et comment les sockets Web et Socket.IO sont des options plus efficaces et sophistiquées que la communication client-serveur classique.

Créer un nouveau projet

Commencez par créer un répertoire nommé **chat-socketio** contenant deux fichiers, l'un nommé **server.js** et l'autre nommé **index.html** :

```
$ mkdir chat-socketio
$ cd chat-socketio
$ touch server.js
$ touch index.html
```

Comme nous allons utiliser un module npm tiers, nous devons également initialiser notre projet :

```
$ npm init --yes
```

Les tâches que notre application effectuera

1. Le serveur enverra le fichier `index.html` sur la route `localhost:3000/`
2. Nous pouvons ouvrir autant d'onglets dans le navigateur sur la route `localhost:3000/` et chaque onglet représente un client.
3. Lorsque nous ouvrons `localhost:3000/`, nous devons définir un nom d'utilisateur. Si le nom d'utilisateur existe, nous recevons un message d'erreur.
4. Lorsqu'un nouvel utilisateur est ajouté, le serveur doit informer tous les onglets ouverts (clients) avec les utilisateurs connectés.
5. Lorsqu'un utilisateur envoie un message dans un onglet, le serveur reçoit le message et l'envoie à tous les clients connectés afin qu'ils puissent tous voir le message.

- Lorsqu'un utilisateur ferme l'onglet, cela signifie qu'il est déconnecté et le serveur mettra à jour la liste des utilisateurs connectés et l'enverra à tous les clients connectés.

Créer le serveur

Dans cet codelab, nous allons créer un serveur WebSocket avec Socket.IO.
Commencez par installer le module **express** et **socket.io** :

```
$ npm install express
```

```
$ npm install socket.io
```

Implémentez le serveur (**server.js**) comme suit :

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

usernames = [];

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

io.sockets.on('connection', function(socket){
  console.log('Socket Connecté...');
  socket.on('new user', function(data, callback){
    if(usernames.indexOf(data) !== -1){
      callback(false);
    } else {
      callback(true);
      socket.username = data;
      usernames.push(socket.username);
      updateUserNames();
    }
  });
  // Mettre à jour les noms d'utilisateur
  function updateUserNames(){
    io.sockets.emit('usernames', usernames);
  }
  // Envoyer le message
  socket.on('send message', function(data){
    io.sockets.emit('new message', {msg: data, user:socket.username});
  });
  // Déconnecter
  socket.on('disconnect', function(data){
    if(!socket.username){
      return;
    }
    usernames.splice(usernames.indexOf(socket.username), 1);
    updateUserNames();
  });
});

server.listen( 3000, ()=>{
  console.log("Le serveur ecoute sur le port 3000");
});
```

liste de tous les utilisateurs connectés

Envoie le fichier index.html lorsqu'une requête HTTP GET arrive à localhost:3000/

lorsqu'un client est connecté (accès localhost:3000)

Lorsqu'un client entre un nom d'utilisateur. Nous vérifions si le nom d'utilisateur existe déjà.

Lorsqu'un client est connecté (et saisi un nom d'utilisateur) ou déconnecté, nous mettons à jour la liste des utilisateurs connectés et l'envoyons à tous les clients connectés en émettant un événement.

Lorsqu'un client envoie un message, le serveur reçoit le message et le diffuse aux clients connectés en émettant un événement.

Lorsqu'un client se déconnecte (ferme l'onglet/ navigateur ouvert), le serveur supprime son nom d'utilisateur de la liste et informe les utilisateurs connectés.

Ouvrez **index.html** et ajoutez ce qui suit (**copier coller**):

```
<!DOCTYPE html>
<html>
<head>
  <title>Appli de Chat</title>
  <style>
    body{
      background: #ceb59f;
    }

    #container{
      width: 800px;
      margin: 0 auto;
      display: flex;
      flex-wrap: wrap;
      justify-content: space-between;
    }

    #chatWindow{
      height: 350px;
      width: 550px;
      border: 2px solid #bcbbc9;
      border-radius: 20px;
      padding: 20px;
      background: #efd1b5;
    }

    #mainWrapper{
      display: none;
    }

    #chatWrapper{
      float:left;
      border:1px #ccc solid;
      border-radius: 10px;
      background:#f4f4f4;
      padding:10px;
    }

    #userWrapper{
      float:left;
      border:1px #ccc solid;
      border-radius: 10px;
      background: #d7b599;
      padding:10px;
      margin-left:20px;
      width:150px;
      max-height:200px;
    }
  </style>
</head>
</html>
```

```

#namesWrapper{
    float:left;
    border:1px #ccc solid;
    border-radius: 10px;
    background: #dcb699;
    padding:10px;
    margin-left:20px;
}

input{
    height: 30px;
    border: solid 2px #d7d7d7;
    border-radius: 10px;
    padding: 10px;
    margin-top: 10px;
}
</style>

</head>
<body>
<div id="container">
    <div id="namesWrapper">
        <h2>ChatI0</h2>
        <p>Create Username:</p>
        <div id="error"></div>
        <form id="usernameForm">
            <input type="text" size="35" id="username">
            <input type="submit" value="Submit">
        </form>
    </div>

    <div id="mainWrapper">
        <h2>Chat avec Socket.IO</h2>
        <div id="chatWrapper">
            <div id="chatWindow"></div>
            <form id="messageForm">
                <input type="text" size="35" id="message" placeholder="Say
Something...">
                <input type="submit" value="Submit">
            </form>
        </div>

        <div id="userWrapper">
            <div id="users"></div>
        </div>
    </div>
</div>

<script src="http://code.jquery.com/jquery-latest.min.js"></script>
<script src="/socket.io/socket.io.js"></script>

```

```

<script>
  $(function(){
    var socket = io.connect();
    var $messageForm = $('#messageForm');
    var $message = $('#message');
    var $chat = $('#chatWindow');
    var $usernameForm = $('#usernameForm');
    var $users = $('#users');
    var $username = $('#username');
    var $error = $('#error');

    $usernameForm.submit(function(e){
      e.preventDefault();
      socket.emit('new user', $username.val(), function(data){
        if(data){
          $('#namesWrapper').hide();
          $('#mainWrapper').show();
        } else{
          $error.html('Username is taken');
        }
      });
    });

    socket.on('usernames', function(data){
      var html = '';
      for(i = 0; i < data.length; i++){
        html += data[i] + '<br>';
      }
      $users.html(html);
    });

    $messageForm.submit(function(e){
      e.preventDefault();
      socket.emit('send message', $message.val());
      $message.val('');
    });

    socket.on('new message', function(data){
      $chat.append('<strong>'+data.user+'</strong>: '+data.msg+'<br>');
    });
  });
</script>
</body>
</html>

```

```
<script>
  $(function(){
    var socket = io.connect();
    var $messageForm = $('#messageForm');
    var $message = $('#message');
    var $chat = $('#chatWindow');
    var $usernameForm = $('#usernameForm');
    var $users = $('#users');
    var $username = $('#username');
    var $error = $('#error');

    $usernameForm.submit(function(e){
      e.preventDefault();
      socket.emit('new user', $username.val(), function(data){
        if(data){
          $('#namesWrapper').hide();
          $('#mainWrapper').show();
        } else{
          $error.html('Username is taken');
        }
      });
    });

    socket.on('usernames', function(data){
      var html = '';
      for(i = 0; i < data.length; i++){
        html += data[i] + '<br>';
      }
      $users.html(html);
    });

    $messageForm.submit(function(e){
      e.preventDefault();
      socket.emit('send message', $message.val());
      $message.val('');
    });

    socket.on('new message', function(data){
      $chat.append('<strong>'+data.user+'</strong>: '+data.msg+'<br>');
    });
  });
</script>
```

Lorsqu'un utilisateur entre son nom d'utilisateur, le client émet un événement et envoie le nom d'utilisateur au serveur. Le serveur renverra un rappel avec une valeur booléenne. Si le nom d'utilisateur est pris, un message d'erreur s'affichera.

Écoute de l'événement "usernames". C'est à ce moment que le serveur informe tous les clients connectés avec la liste mise à jour des utilisateurs connectés.

Envoi d'un message à tous les clients connectés (via le serveur)

Mise à jour de la boîte de discussion / chat box (un nouveau message a été reçu)

Maintenant, démarrez votre serveur dans une fenêtre Terminal :

```
$ node server.js
```

Ouvrez de nombreux onglets dans votre navigateur avec l'url **localhost:3000/** et testez votre application. Dans chaque onglet, vous pouvez créer un nouvel utilisateur. Lorsque vous envoyez un message dans un onglet, le message doit apparaître dans tous les autres. Vous devriez également pouvoir voir la liste de tous les utilisateurs connectés. Lorsque vous fermez un onglet, l'utilisateur doit disparaître de la liste des utilisateurs connectés.

ChatIO

Create Username:

Submit

Chat avec Socket.IO

Submit

David
Joseph

Chat avec Socket.IO

Joseph: Bonjour
David: Salut!!!

Submit

David
Joseph

Comment aller plus loin ?

- Vous ajoutez un horodatage à chaque message.
- Vous créez un modèle de données pour vos messages de chat et chargez les messages de la base de données lorsque vous ouvrez la page de chat de l'application.
- Vous implémentez une icône dans la barre de navigation qui agit comme un indicateur lorsque la page de chat est active, même lorsque l'utilisateur est sur une autre page.
- Soyez créatif et réfléchissez à la manière dont vous pouvez intégrer Socket.IO dans un projet existant...



Communication full duplex et à faible latence pour toutes les plateformes

Tutoriel

Documentation

<https://socket.io/fr/>



Performant

Dans la majorité des cas, la connexion sera établie avec des WebSockets, fournissant un canal de communication à faible surcharge entre le serveur et le client.



Fiable

Vous pouvez être sereins ! Dans les rares cas où la connexion WebSocket n'est pas possible, le client utilisera du HTTP long-polling. Et si la connexion est interrompue, le client essaiera automatiquement de se reconnecter.



Évolutif

Déployez votre application sur plusieurs serveurs et envoyez facilement des événements à tous les clients connectés.

Exemple de base

```
import { Server } from "socket.io";
const io = new Server(3000);

io.on("connection", (socket) => {
  // send a message to the client
  socket.emit("hello", "world");

  // receive a message from the client
  socket.on("howdy", (arg) => {
    console.log(arg); // prints "stranger"
  });
});
```

```
import { io } from "socket.io-client";
const socket = io("ws://localhost:3000");

// receive a message from the server
socket.on("hello", (arg) => {
  console.log(arg); // prints "world"
});

// send a message to the server
socket.emit("howdy", "stranger");
```