

Desarrollar un chat con socket.io

Sonsoles López Pernas

Pasos

1. Instalar las dependencias
2. Crear el servidor web
3. Crear el cliente web
4. Añadir el servidor de socket.io al servidor web
5. Añadir el cliente de socket.io al cliente web
6. Lograr comunicación bidireccional cliente-servidor
7. Crear la interfaz del chat y enviar mensajes
8. Reenviar los mensajes del chat en el servidor
9. Recibir los mensajes en el chat
10. Añadir nombres de usuario
11. Crear avisos cuando alguien se conecta
12. Crear avisos cuando alguien se desconecta

Paso 1: Instalar las dependencias

El primer paso para desarrollar un servidor con node.js y Websockets será crear el fichero package.json :

package.json

```
{
  "name": "chat",
  "version": "1.0.0",
  "description": "Websocket chat example",
  "scripts": {
    "start": "node index.js"
  },
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1",
    "socket.io": "^2.2.0"
  }
}
```

En el mismo directorio donde hemos creado el fichero package.json ejecutamos `npm install` para instalar las dependencias indicadas en el package.json.

Paso 2: Crear el servidor web

Para crear el servidor web utilizamos el framework express.
En el directorio de trabajo creamos un nuevo fichero
llamado `index.js` :

`index.js`

```
var express = require('express');  
var app = express();  
var http = require('http').Server(app);  
var path = require('path');  
var port = process.env.PORT || 3000;
```

```
app.use(express.static(path.join(__dirname, 'public')));  
app.get('*', function(req, res, next) {  
  res.send("Hello");  
});
```

El servidor servirá
los ficheros situados
en la carpeta public

```
http.listen(port, function() {  
  console.log('Open your browser on http://localhost:' + port);  
});
```

El servidor escucha
en el puerto 3000

Para arrancar el servidor ejecutamos `npm start`


Si abrimos el navegador en la página <http://localhost:3000>
veremos el mensaje “Hello!”

Paso 3: Crear el cliente web

Para alojar el cliente web creamos un directorio llamado `public` y dentro un fichero `index.html` con el siguiente contenido:

`index.html`

```
<!doctype html>
<html>
  <head>
    <title>Socket.I O chat</title>
    <script src="https://code.jquery.com/jquery-1.11.1.js"></script>
  </head>
  <body>
    M chat con websockets
  </body>
</html>
```



A purple L-shaped bracket connects the `src="https://code.jquery.com/jquery-1.11.1.js"` attribute in the `<script>` tag to the text "Importamos jQuery".

Si refrescamos la página <http://localhost:3000> en el navegador veremos que se muestra el mensaje “Mi chat con websockets”.

Paso 4: Añadir servidor de socket.io

Con el código que hemos incluido hasta ahora, tenemos un servidor node.js sencillo que sirve ficheros. El siguiente paso es añadir el servidor de socket.io para poder establecer comunicación bidireccional entre cliente y servidor.

Primero, en el fichero `index.js` añadimos la dependencia en la parte superior del fichero (después de importar el módulo `http`) e inicializamos el servidor de sockets.

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

Al final del fichero escuchamos cada nueva conexión:

```
io.on('connection', function(socket) {  
    console.log("New connection");  
});
```

Paso 5: Añadir cliente de socket.io

Para poder conectarnos al servidor de socket.io, tenemos que crear un cliente. Primero, necesitamos incluir la librería de socket.io-client en nuestro `index.html`. Al final de la cabecera (`head`) añadimos:

```
<script
  src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.2.0/socket.io.dev.js">
</script>
```

A continuación, creamos la conexión con el servidor de socket.io. Dentro del `body` añadimos:

```
<script>
  $(function () {
    const socket = io.connect("http://localhost:3000");
    socket.on("connect", function() {
      alert("Connected!");
    });
  });
</script>
```

Paso 6: Lograr comunicación bidireccional

Para probarlo debemos reiniciar el servidor (deteniéndolo y volviendo a arrancar con `npm start`) y recargar la página.

- En el terminal del servidor veremos que ha habido una nueva conexión: “New connection”
- En el navegador aparecerá un mensaje confirmando la conexión “Connected!”

Paso 7: Crear la interfaz del chat

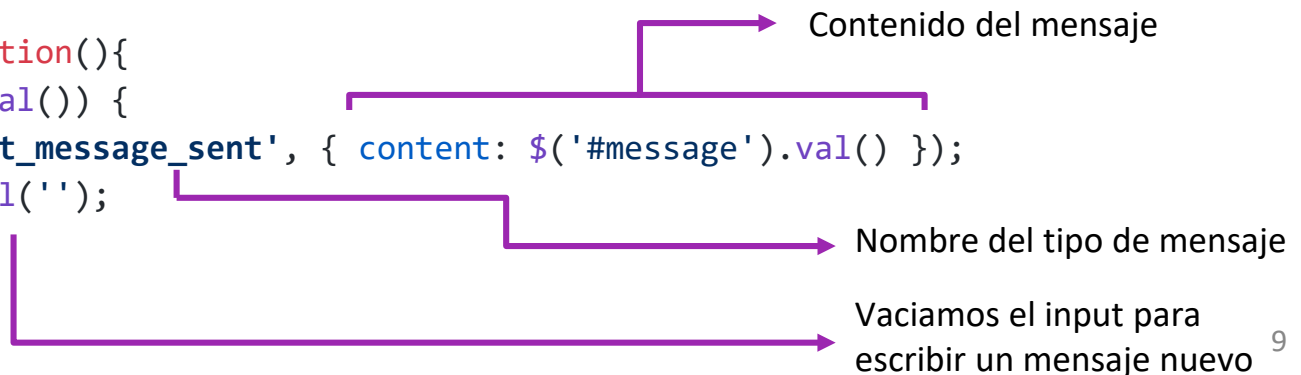
Creamos la interfaz del chat en el HTML. Para ello sustituimos el texto del `body` por las siguientes líneas:

```
<ul id="messages"></ul>
<form action="">
  <input id="message" autocomplete="off" placeholder="Start writing..." />
  <button id="send" type="submit" title="Send a message" >
    Send
  </button>
</form>
```



Al pulsar el botón “Send” debemos enviar el mensaje escrito en el `input` de texto. Para ello añadimos el siguiente código dentro de la etiqueta `script`:

```
$( 'form' ).submit( function() {
  if ( $( '#message' ).val() ) {
    socket.emit( 'chat_message_sent', { content: $( '#message' ).val() } );
    $( '#message' ).val( '' );
  }
  return false;
} );
```



Contenido del mensaje

Nombre del tipo de mensaje

Vaciamos el input para escribir un mensaje nuevo

Paso 8: Reenviar los mensajes en el servidor

Para que a todos los participantes del chat les llegue el mensaje que hemos enviado debemos recogerlo en el servidor y re-enviárselo a todos los conectados.

Para ello, debemos escuchar los mensajes con el mismo nombre con el que los enviamos desde el cliente.

```
io.on('connection', function(socket) {  
  console.log("New connection");  
  socket.on("chat_message_sent", function(msg) {  
    io.emit("chat_message_received", msg);  
  });  
});
```

Reenviamos el mensaje a todos usando un nuevo tipo de mensaje "chat_message_received".

Paso 9: Recibir mensajes en el chat

Para recibir en el cliente los mensajes y mostrarlos en la interfaz de chat los mensajes añadimos el siguiente código en el cliente:

```
socket . on( " chat_message_received ", function( msg ) {  
    $( "#messages" ). append( ` <li>${ msg. content } </li> ` );  
});
```

Escucha el mensaje
con ese nombre

Muestra el mensaje
en el chat

Si abrimos la página <http://localhost:3000> en varias pestañas del navegador, podemos escribir mensajes en la caja de texto y se verán en forma de lista en todas las pestañas.

Paso 10: Añadir nombres de usuario

Para saber quién manda cada mensaje, podemos preguntar a cada usuario su nombre.

Antes de inicializar la conexión al servidor de sockets, añadimos en la web la siguiente línea:

```
const name = prompt("What is your name?");
```

Cada vez que mandamos un mensaje, añadimos el nombre del usuario que lo envía.

```
socket.emit('chat_message_sent',  
    { content: $('#message').val(), user: name });
```

Cuando recibimos un mensaje, mostramos quién lo ha mandado:

```
$("#messages").append(  
    `<li>${msg.user}: ${msg.content}</li>`);
```

Paso 11: Crear aviso de nuevo participante

Para avisar a todos los participantes de que alguien nuevo se ha conectado al chat, podemos lanzar un nuevo mensaje cada vez que haya una nueva conexión en el servidor:

```
io.on('connection', function(socket){
  console.log(" New connection");
  io . emit ( "new_member" );
  socket . on( " chat_message_sent ", function( msg ) {
    io . emit ( " chat_message_received ", msg );
  });
});
```

Tenemos que recibirlo en el cliente y avisar a todos de la nueva incorporación.

```
socket . on( 'new_member' , function( msg ) {
  $( '#messages' ) . append( `<li class="announcement">
    Someone has joined the conversation</li>` );
});
```

Paso 12: Crear aviso de desconexión

Cuando un cliente se cierra, el servidor recibe un evento de desconexión automáticamente. Para avisar a todos los participantes de que alguien ha salido del chat, podemos escuchar el evento de desconexión y enviar un nuevo mensaje a todos en el caso de que éste tenga lugar.

```
socket . on( 'disconnect' , function( msg) {  
    io. emit( 'member_exit' );  
});
```

En el cliente, si recibimos un mensaje del tipo “`member_exit`”, creamos un nuevo anuncio para notificar a los participantes restantes:

```
socket . on( 'member_exit' , function( msg) {  
    $( '#messages' ). append( `<li class="announcement">  
        Someone has left the conversation</li>` );  
});
```

Resultado final

Si reiniciamos de nuevo el servidor y abrimos la página <http://localhost:3000> en varias pestañas podemos comprobar que funcionan todas las funcionalidades que hemos desarrollado:

- Someone has joined the conversation
- user1: hello!
- user2: :)
- Someone has left the conversation

Desarrollar un chat con socket.io

Sonsoles López Pernas
