***Creación de chat básico***

***EL framework web***

Primero pasos para chat básico. (nota: cada vez que creemos un proyecto nuevo habrá que realizar estos pasos)

Instalación node.js:

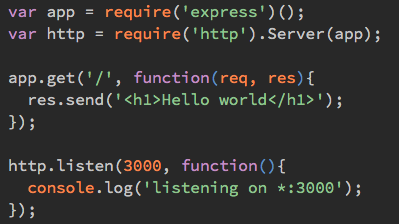
-Crear una carpeta (ej.chat basico)

-Irse al terminal y meterse en dicha carpeta (cd chat basico)

-Introducir el comando npm init

-Nos pide los datos para crear el JSON

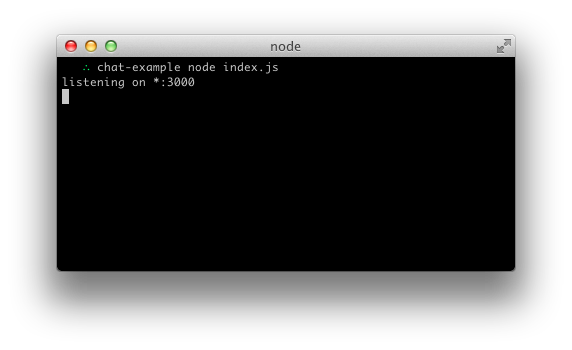
-Introducir el comando npm install express para instalar express que es un framework de node.js que proporciona funcionalidades como enrutamiento, Opciones para gestionar sesiones y cookies…



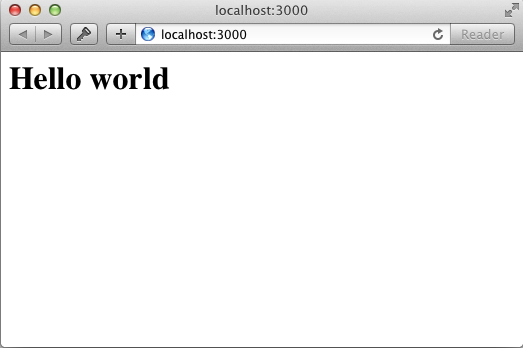
Esto se traduce en lo siguiente:

* Express se inicializa app para ser un controlador de funciones que puede suministrar a un servidor HTTP (como se ve en la línea 2).
* Definimos un controlador de ruta /que recibe llamadas cuando ingresamos a nuestro sitio web.
* Hacemos que el servidor http escuche en el puerto 3000.

Si ejecuta node index.js debería ver lo siguiente:

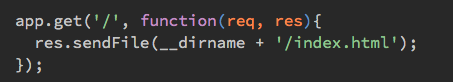


And if you point your browser to <http://localhost:3000>:



## *Serving HTML*

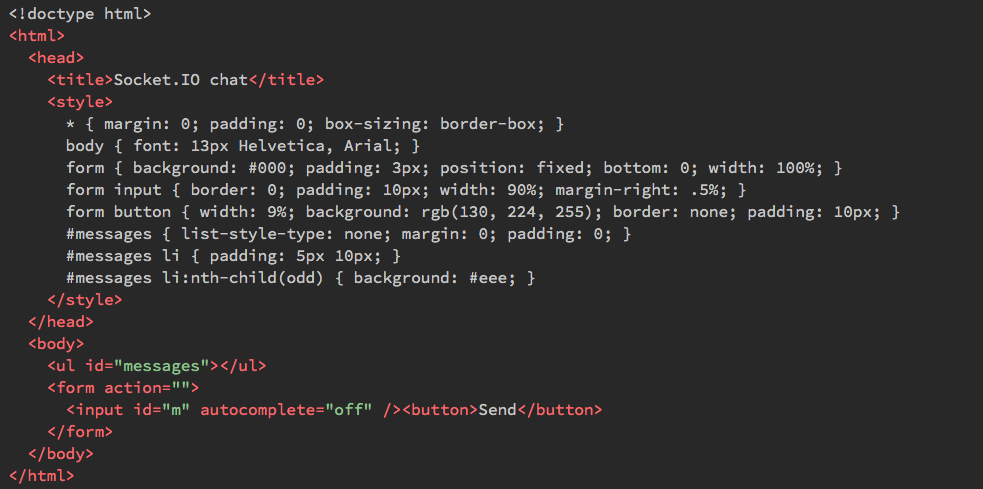
Hasta ahora index.js estamos llamando res.send y le pasamos una cadena HTML. Nuestro código sería muy confuso si pusiéramos allí el código HTML de toda nuestra aplicación. En su lugar, vamos a crear un index.html archivo y servirlo.

Vamos a re factorizar nuestro controlador de ruta para usar sendFile en su lugar: 

|  |
| --- |
|  |

Y rellene index.html con lo siguiente:

|  |
| --- |
|  |



Si reinicia el proceso (presionando Control + C y node index vuelve a ejecutar) y actualiza la página, debería tener este aspecto:



## *Integración Socket.IO*

Socket.IO se compone de dos partes:

* Un servidor que se integra con (o se monta en) el servidor HTTP [Node.JS](https://github.com/socketio/socket.io) : [socket.io](https://github.com/socketio/socket.io)
* Una biblioteca cliente que se carga en el navegador: [socket.io-client](https://github.com/socketio/socket.io-client)

Durante el desarrollo, socket.io el cliente nos sirve automáticamente, como veremos, así que por ahora solo tenemos que instalar un módulo:

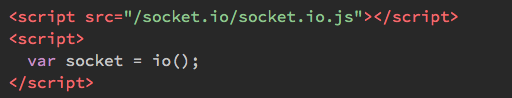
|  |
| --- |
| npm instalar --save socket.io |

Eso instalará el módulo y agregará la dependencia a package.json. Ahora vamos a editar index.js para agregarlo:



Observe que inicializo una nueva instancia socket.io pasando el objeto http (el servidor HTTP). Luego escucho en el connection evento los sockets entrantes y lo registro en la consola.

Ahora en index.html agrego el siguiente fragmento de código antes de </body>:

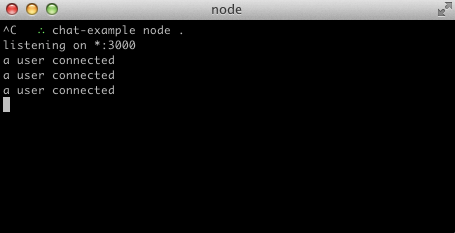


Eso es todo lo que se necesita para cargar el socket.io-client, que expone un io global, y luego conectarse.

Tenga en cuenta que no estoy especificando ninguna URL cuando llamo io (), ya que por defecto intenta conectar al host que sirve la página.

Si ahora vuelve a cargar el servidor y el sitio web, debería ver la impresión de la consola "un usuario conectado".

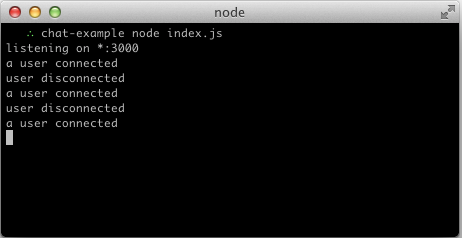
Intenta abrir varias pestañas y verás varios mensajes:



Cada toma también dispara un disconnect evento especial:



Luego, si actualiza una pestaña varias veces, puede verla en acción:



## *Emitiendo eventos*

La idea principal detrás de Socket.IO es que puede enviar y recibir cualquier evento que desee, con la información que desee. Cualquier objeto que pueda codificarse como JSON funcionará, y [los datos binarios también](https://socket.io/blog/introducing-socket-io-1-0/#binary) son compatibles.

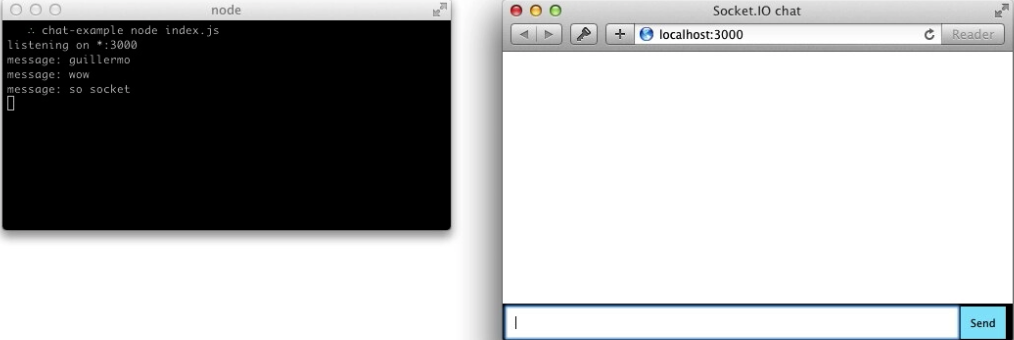
Hagámoslo de modo que cuando el usuario escriba un mensaje, el servidor lo reciba como un chat message evento. La script sección s en index.html ahora debería verse como sigue:



Y en index.js que imprimimos el chat message evento:



El resultado debería ser como el siguiente video:



## *Radiodifusión*

El siguiente objetivo es que emitamos el evento desde el servidor al resto de usuarios.

Para enviar un evento a todos, Socket.IO nos ofrece io.emit:

En este caso, por simplicidad, enviaremos el mensaje a todos, incluido el remitente.



Y en el lado del cliente, cuando capturemos un chat message evento, lo incluiremos en la página. El código total de JavaScript del lado del cliente ahora equivale a:



¡Y eso completa nuestra aplicación de chat, en aproximadamente 20 líneas de código! Esto es lo que parece:

