



**EBook Gratis**

# APRENDIZAJE socket.io

Free unaffiliated eBook created from  
**Stack Overflow contributors.**

**#socket.io**

# Tabla de contenido

<b>Acerca de</b> .....	<b>1</b>
<b>Capítulo 1: Empezando con socket.io</b> .....	<b>2</b>
Observaciones.....	2
Versiones.....	2
Examples.....	3
Instalación o configuración.....	3
"¡Hola Mundo!" Con mensajes de socket.....	4
<b>Capítulo 2: Emisión</b> .....	<b>6</b>
Examples.....	6
Difusión a todos los usuarios.....	6
Transmitir a todos los otros sockets.....	6
<b>Capítulo 3: Escuchar eventos</b> .....	<b>7</b>
Examples.....	7
Escuchando eventos internos y personalizados:.....	7
<b>Capítulo 4: Eventos de fuego</b> .....	<b>8</b>
Examples.....	8
Eventos personalizados de fuego.....	8
<b>Capítulo 5: Manejo de usuarios con socket.io</b> .....	<b>9</b>
Introducción.....	9
Examples.....	9
Ejemplo de código del lado del servidor para manejar usuarios.....	9
Manera simple de emitir mensajes por identificación de usuario.....	11
Manejo de usuarios accediendo a los modales.....	11
<b>Creditos</b> .....	<b>13</b>

---

## Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [socket-io](#)

It is an unofficial and free socket.io ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official socket.io.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to [info@zzzprojects.com](mailto:info@zzzprojects.com)

---

# Capítulo 1: Empezando con socket.io

## Observaciones

`Socket.IO` es una biblioteca de JavaScript para aplicaciones web en `realtime`. Permite la comunicación bidireccional en tiempo real entre clientes web y servidores. Tiene dos partes: una biblioteca del lado del cliente que se ejecuta en el navegador y una biblioteca del lado del servidor para `node.js`. Ambos componentes tienen una API casi idéntica. Al igual que `node.js`, está controlado por eventos.

`Socket.IO` utiliza principalmente el protocolo `websocket` con el sondeo como una opción alternativa, al tiempo que proporciona la misma interfaz. Aunque se puede usar como un simple envoltorio para `WebSocket`, proporciona muchas más funciones, incluida la difusión a múltiples sockets, el almacenamiento de datos asociados con cada cliente y la E / S asíncrona.

## Versiones

Versión	Fecha de lanzamiento
1.4.8	2016-06-23
1.4.7	2016-06-23
1.4.6	2016-05-02
1.4.5	2016-01-26
1.4.4	2016-01-10
1.4.3	2016-01-08
1.4.2	2016-01-07
1.4.1	2016-01-07
1.4.0	2015-11-28
1.3.7	2015-09-21
1.3.6	2015-07-14
1.3.5	2015-03-03
1.3.4	2015-02-14
1.3.3	2015-02-03

Versión	Fecha de lanzamiento
1.3.2	2015-01-19
1.3.1	2015-01-19
1.3.0	2015-01-19
1.2.1	2014-11-21
1.2.0	2014-10-27
1.1.0	2014-09-04
1.0.6	2014-06-19
1.0.5	2014-06-16
1.0.4	2014-06-02
1.0.3	2014-05-31
1.0.2	2014-05-28
1.0.1	2014-05-28
1.0.0	2014-05-28

## Examples

### Instalación o configuración

En primer lugar, instale `socket.io` módulo en `node.js` aplicación.

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

### Configuración básica de HTTP

El siguiente ejemplo conecta `socket.io` a un servidor HTTP de `node.js` simple que escucha en el puerto 3000.

```
var server = require('http').createServer();

var io = require('socket.io')(server);

io.on('connection', function(socket){

  console.log('user connected with socketId '+socket.id);

  socket.on('event', function(data){
    console.log('event fired');
  });
});
```

```

});

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

});

server.listen(3000);

```

## Configuración con Express

La aplicación Express se puede pasar al servidor `http` que se adjuntará a `socket.io`.

```

var app = require('express')(); //express app
var server = require('http').createServer(app); //passed to http server
var io = require('socket.io')(server); //http server passed to socket.io

io.on('connection', function(){

  console.log('user connected with socketId '+socket.id);

  socket.on('event', function(data){
    console.log('event fired');
  });

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

});

server.listen(3000);

```

## Configuración del lado del cliente

Verifique el ejemplo anterior de Hello World para la implementación del lado del cliente.

## "¡Hola Mundo!" Con mensajes de socket.

### Instalar módulos de nodo

```

npm install express
npm install socket.io

```

### Servidor Node.js

```

const express = require('express');
const app = express();
const server = app.listen(3000, console.log("Socket.io Hello Wolrd server started!"));
const io = require('socket.io')(server);

io.on('connection', (socket) => {
  //console.log("Client connected!");
  socket.on('message-from-client-to-server', (msg) => {

```

```
        console.log(msg);
    })
    socket.emit('message-from-server-to-client', 'Hello World!');
});
```

## Cliente navegador

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Hello World with Socket.io</title>
  </head>
  <body>
    <script src="https://cdn.socket.io/socket.io-1.4.5.js"></script>
    <script>
      var socket = io("http://localhost:3000");
      socket.on("message-from-server-to-client", function(msg) {
        document.getElementById('message').innerHTML = msg;
      });
      socket.emit('message-from-client-to-server', 'Hello World!');
    </script>
    <p>Socket.io Hello World client started!</p>
    <p id="message"></p>
  </body>
</html>
```

En el ejemplo anterior, la ruta a la biblioteca socket.io se define como `/socket.io/socket.io.js`.

Aunque no escribimos ningún código para servir a la biblioteca socket.io, Socket.io lo hace automáticamente.

Lea [Empezando con socket.io en línea](https://riptutorial.com/es/socket-io/topic/3588/empezando-con-socket-io): <https://riptutorial.com/es/socket-io/topic/3588/empezando-con-socket-io>

---

# Capítulo 2: Emisión

## Examples

### Difusión a todos los usuarios.

Es posible enviar un mensaje o datos a todas las conexiones disponibles. Esto se puede lograr inicializando primero el servidor y luego utilizando el objeto socket.io para encontrar todos los sockets y luego emitirlos como lo haría normalmente con un solo socket

```
var io = require('socket.io')(80) // 80 is the HTTP port
io.on('connection', function (socket) {
  //Callback when a socket connects
  });
io.sockets.emit('callbackFunction',data);
```

### Transmitir a todos los otros sockets

Es posible emitir un mensaje o datos a todos los usuarios, excepto al que realiza la solicitud:

```
var io = require('socket.io')(80);
io.on('connection', function (socket) {
  socket.broadcast.emit('user connected');
});
```

Lea Emisión en línea: <https://riptutorial.com/es/socket-io/topic/6295/emision>



---

# Capítulo 3: Escuchar eventos

## Examples

### Escuchando eventos internos y personalizados:

#### Server Syntax

```
var io = require('socket.io')(80);

io.on('connection', function (mysocket) {

  //custom event called `private message`
  mysocket.on('private message', function (from, msg) {
    console.log('I received a private message by ', from, ' saying ', msg);
  });

  //internal `disconnect` event fired, when a socket disconnects
  mysocket.on('disconnect', function () {
    console.log('user disconnected');
  });
});
```

#### Client syntax :

```
var mysocket = io('http://example.com');
mysocket.on('private message', function (data) {
  console.log(data);
});
```

Lea Escuchar eventos en línea: <https://riptutorial.com/es/socket-io/topic/4455/escuchar-eventos>

---

# Capítulo 4: Eventos de fuego

## Examples

### Eventos personalizados de fuego

#### Server syntax :

```
var io = require('socket.io')(80);
io.on('connection', function (mysocket) {
  //emit to all but the one who started it
  mysocket.broadcast.emit('user connected');

  //emit to all sockets
  io.emit('my event', { messg: 'for all' });
});

// a javascript client would listen like this:
// var mysocket = io('http://example.com');
// mysocket.on('my event', function (data) {
//   console.log(data);
// });
```

#### Client syntax :

```
var mysocket = io('http://example.com');
mysocket.emit('another event', { messg: 'hello' });

// a node.js server would listen like this:
// require('socket.io')(80).on('connection', function (mysocket) {
//   mysocket.on('another event', function (data) {
//     console.log('data from client : '+ data);
//   });
// });
```

Lea Eventos de fuego en línea: <https://riptutorial.com/es/socket-io/topic/6625/eventos-de-fuego>

---

# Capítulo 5: Manejo de usuarios con socket.io

## Introducción

El manejo de usuarios dentro de socket.io es tan simple o tan complejo como decidió, aunque hay algunos enfoques más "obvios" para hacer esto, esta documentación va a describir un enfoque utilizando `map()`.

## Examples

### Ejemplo de código del lado del servidor para manejar usuarios

En primer lugar, es importante tener en cuenta que cuando se crea un nuevo socket, se le asigna un `id`. Único que se recupera llamando a `socket.id`. Esta `id` se puede almacenar dentro de un objeto de `user` y podemos asignar un identificador como un nombre de usuario que se ha utilizado en este ejemplo para recuperar objetos de `user`.

```
/**
 * Created by Liam Read on 27/04/2017.
 */

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

function User(socketId) {

  this.id = socketId;
  this.status = "online";
  this.username = "bob";

  this.getId = function () {
    return this.id;
  };

  this.getName = function () {
    return this.username;
  };

  this.getStatus = function () {
    return this.status;
  };

  this.setStatus = function (newStatus) {
    this.status = newStatus;
  }
}

var userMap = new Map();

/**
```

```

* Once a connection has been opened this will be called.
*/
io.on('connection', function (socket) {

  var user;

  /**
   * When a user has entered there username and password we create a new entry within the
userMap.
   */
  socket.on('registerUser', function (data) {

    userMap.set(data.name, new User(socket.id));

    //Lets make the user object available to all other methods to make our code DRY.
    user = userMap.get(data.name);
  });

  socket.on('loginUser', function (data) {
    if (userMap.has(data.name)) {
      //user has been found

      user = userMap.get(data.name);
    } else {
      //Let the client know that no account was found when attempting to sign in.
      socket.emit('noAccountFound', {
        msg: "No account was found"
      });
    }
  });

  socket.on('disconnect', function () {
    //Let's set this users status to offline.
    user.setStatus("offline");
  });

  /**
   * Dummy server event that represents a client looking to send a message to another user.
   */
  socket.on('sendAnotherUserAMessage', function (data) {

    //Make note here that by checking to see if the user exists within the map we can be
sure that when
    // retrieving the value after && that we won't have any unexpected errors.
    if (userMap.has(data.name) && userMap.get(data.name).getStatus() !== "offline") {
      var OtherUser = userMap.get(data.name);
    } else {
      //We use a return here so further code isn't executed, you could replace this with
some for of
      //error handling or a different event back to the user.
      return;
    }

    //Lets send our message to the user.
    io.to(OtherUser.getId()).emit('recMessage', {
      msg: "Nice code!"
    });
  });

});

```

Esto no es de ninguna manera un ejemplo completo de lo que es posible, sino que debe proporcionar una comprensión básica de un enfoque para el manejo de los `users` .

## Manera simple de emitir mensajes por identificación de usuario

En el servidor:

```
var express = require('express');
var socketio = require('socket.io');

var app = express();
var server = http.createServer(app);
var io = socketio(server);

io.on('connect', function (socket) {
  socket.on('userConnected', socket.join);
  socket.on('userDisconnected', socket.leave);
});

function message (userId, event, data) {
  io.sockets.to(userId).emit(event, data);
}
```

En el cliente:

```
var socket = io('http://localhost:9000'); // Server endpoint

socket.on('connect', connectUser);

socket.on('message', function (data) {
  console.log(data);
});

function connectUser () { // Called whenever a user signs in
  var userId = ... // Retrieve userId
  if (!userId) return;
  socket.emit('userConnected', userId);
}

function disconnectUser () { // Called whenever a user signs out
  var userId = ... // Retrieve userId
  if (!userId) return;
  socket.emit('userDisconnected', userId);
}
```

Este método permite enviar mensajes a usuarios específicos por ID única sin tener una referencia a todos los sockets en el servidor.

## Manejo de usuarios accediendo a los modales.

Este ejemplo muestra cómo podría manejar a los usuarios que interactúan con los modales en una base 1-1.

```
//client side
function modals(socket) {
```

```

this.sendModalOpen = (modalIdentifier) => {
    socket.emit('openedModal', {
        modal: modalIdentifier
    });
};

this.closeModal = () => {
    socket.emit('closedModal', {
        modal: modalIdentifier
    });
};
}

socket.on('recModalInfo', (data) => {
    for (let x = 0; x < data.info.length; x++) {
        console.log(data.info[x][0] + " has open " + data.info[x][1]);
    }
});

//server side
let modal = new Map();

io.on('connection', (socket) => {
    //Here we are sending any new connections a list of all current modals being viewed with
    Identifiers.
    //You could send all of the items inside the map() using map.entries

    let currentInfo = [];

    modal.forEach((value, key) => {
        currentInfo.push([key, value]);
    });

    socket.emit('recModalInfo', {
        info: currentInfo
    });

    socket.on('openedModal', (data) => {
        modal.set(socket.id, data.modalIdentifier);
    });

    socket.on('closedModal', (data) => {
        modal.delete(socket.id);
    });
});
});

```

Al manejar todas las interacciones modales aquí, todos los usuarios recién conectados tendrán toda la información sobre cuáles se están viendo actualmente, lo que nos permite tomar decisiones basadas en los usuarios actuales de nuestro sistema.

Lea Manejo de usuarios con socket.io en línea: <https://riptutorial.com/es/socket-io/topic/9837/manejo-de-usuarios-con-socket-io>

# Creditos

S. No	Capítulos	Contributors
1	Empezando con socket.io	<a href="#">Blubberguy22</a> , <a href="#">Cerbrus</a> , <a href="#">Community</a> , <a href="#">Forivin</a> , <a href="#">Iceman</a> , <a href="#">Mohit Gangrade</a> , <a href="#">Mukesh Sharma</a>
2	Emisión	<a href="#">Delapouite</a> , <a href="#">Marc Rasmussen</a>
3	Escuchar eventos	<a href="#">Iceman</a>
4	Eventos de fuego	<a href="#">Florian Hämmerle</a> , <a href="#">Iceman</a>
5	Manejo de usuarios con socket.io	<a href="#">li x</a> , <a href="#">Sky</a>