Socket.IO

Make a realtime web chat application in Node.js

What is it

Socket.IO is a library for building WebSocket NodeJS application.

It ships with client-side library that provides an APIs for developing the browser part of the application.

Socket.IO is composed of two parts:

A server that integrates with the Node.JS HTTP Server: `socket.io`

A client library that loads on the browser side: `socket.io-client`

What it did:

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.

Install

npm install socket.io

References

Create a simple Chat:

https://socket.io/get-started/chat/

Professional Node Book:

Local LINK Page: 275-300

Website Documentation:

https://socket.io/docs

Created by: Wael Zoaiter - Github

Initialize

Server-Side

```
1   var app = require('express')();
2   var http = require('http').Server(app);
3   var io = require('socket.io')(http);
4
5   // Routes Handler
6   app.get('/',function(req,res){
7       res.sendFile(__dirname + '/index.html');
8   });
9
10  // Socket.IO connection
11  io.on('connection',function(socket){
12      console.log('User Connected');
13      socket.on('disconnect', function(){
14      console.log('User Disconnected');
15    });
16  });
17
18  // Listen to server
19  http.listen(3000,function(){
20      console.log('listening on port 3000');
21  });
```

#3: initialize a new instance "new socket" of `socket.io` by passing the `http` (the HTTP server) object.

#11: listen on the `connection` event for incoming sockets, and log it to the console.

And every time someone open the host or refresh it, we can see it in action:

```
[nodemon] starting `node index.js`
listening on port 3000
User Connected
User Disconnected
User Connected
```

Client-Side

```
18    <body>
19      <ul id="messages"></ul>
20      <form action="">
21        <input id="m" autocomplete="off" /><button>Send</button>
22      </form>
23      <script src="/socket.io/socket.io.js"></script>
24      <script>
25        var socket = io();
26      </script>
27    </body>
```

#23: load the `socket.io-client`, which exposes a `io` global.

#25: `io()` it defaults to trying to connect to the host that serves the page.

#25 : Make a connection to the server .. Listening to the server .. connection between the client and the server.

Event emitter

Client-Side

```
23      <script src="/socket.io/socket.io.js"></script>
24      <script src="https://code.jquery.com/jquery-1.11.1
25      <script>
26        $(function () {
27          var socket = io();
28          $('form').submit(function(){
29            socket.emit('chat message', $('#m').val());
30            $('#m').val('');
31            return false;
32          });
33          socket.on('chat message', function(msg){
34            $('#messages').append($('<li>').text(msg));
35          });
36        });
37      </script>
```

#28: When 'Submit' event happen on the 'form', execute the function:

#29: invoke '.emit()' function with event 'chat message' and its data.

#33: When receiving a data from 'chat message event, fire this function:

That print the event data.

Server-Side

```
10  // Socket.IO connection
11  io.on('connection',function(socket){
12      // console.log('User Connected');
13      socket.on('chat message',function(msg){
14          //console.log('message: ' + msg);
15          io.emit('chat message',msg);
16      });
17      // socket.on('disconnect', function(){
18      //  console.log('User Disconnected');
19      // });
20  });
```

#11: When connection event occurs, performe the function :

#13: When the server **receives** a 'chat message' event, he'll perform the function that have the event **data** 'msg'.

#15: **Send** a **data** 'msg' to the 'chat message' event everywhere.

Emit/Receive Data

Client-Side

```
27      socket.emit('private message');
```

Emitting the event to execute it in the server-side.

Server-Side

```
23      socket.on('private message', function(from,msg){
24          console.log('Private message have been
                received from ' + from + ' that said ' +
                 msg);
25      });
```

CMD:

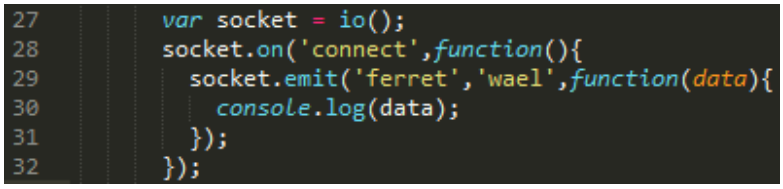
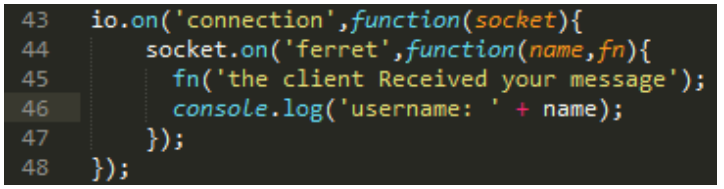
```
listening on port 3000
Private message have been received from undefinedthat said undefined
```

Check message / Sender name

When you want to get a callback when the client confirmed the message reception,

You can pass a function as a last parameter of `.emit` or `.send` .

Created by: Wael Zoaiter - Github

Client-Side	Server-Side
	
#29: Emit ‘ferret’ event with the name of sender and callback function of the data we’ll get.	**#44:** when we receive the ‘ferret’ event we fire function ‘fn’ to the console that we received the event. **#46:** Print the name of sender.

Created by: Wael Zoaiter - Github

Namespace

Namespace is a way to send event from/to special namespace in the client and server side.

So in the **`server-side`** when I emit an event in specific namespace '/group1', I will receive this event only in this namespace in the **`client-side`**.

Important Notes:

- ✓ Which means (assigning different endpoints).
- ✓ introducing separation between communication channels.
- ✓ The client connects to the namespace and the server listen to it.
- ✓ The namespace identified by `io`.
- ✓ Each namespace emits a `connection` event.

Client-Side

Index.html

```
27      // Namespace
28      var chat = io('http://localhost:3000/chat');
29      chat.on('message', function(data) {
30        console.log(data);
31      });
32
33      var news = io('/news');
34      news.on('hello',function(data){
35        console.log('Event reseived');
36      });
```

#28&33: Define the same namespace to connect to the namespace.

The namespace on the server side emit the events 'message & hello' and we are here in the client side receiving them.

Server-Side

Index.js

```
30  // Namespace
31  var chat = io.of('/chat');
32  chat.on('connection', function(socket){
33          socket.emit('message', {
34              sendto: '/chat only'
35          });
36      });
37
38  var news = io.of('/news');
39  news.on('connection',function(socket){
40      news.emit('hello');
41  });
```

#31: Set-up a namespace '/chat'.

#32: When someone connect this namespace, execute the function.

Using Namespaces

This simple chat application could be a part of a larger website. If that website provided some near-real-time interactions that used Socket.IO, you would need to be careful and separate all server and client event names so they don't clash.

Fortunately, Socket.IO allows you to use namespaces to separate different applications on the server and on the client. First you have to connect to a specific namespaced Socket.IO URL on the client:

```
var socket = io.connect('http://localhost:4000/chat');
```

And then the chat server has to scope the sockets to only those that use this namespace:

```
var chat = io.of('/chat');

chat.on('connection', function (socket) {
  socket.on('clientMessage', function(content) {
  // …
```

Now you can easily add another application to the client and server using the same technique without having to worry about whether the event names clash.

Rooms

Within each namespace we can define a channel, that sockets can `join` and `leave`.

Join and Leave:

We can call `join` function to subscribe the socket to a channel we already defined.

```
io.on('connection', function(socket){
  socket.join('some room');
});
```

Or `leave` in the same way, then use:

To and In: `they are the same`

`to` which room we want to `emit` our event.

```
io.to('some room').emit('some event');
```

Join by ID:

Each socket has an **ID**, and he automatically join a room identified by this **ID**.

```
io.on('connection', function(socket){
  socket.on('say to someone', function(id, msg){
    socket.broadcast.to(id).emit('my message', msg);
  });
});
```

Server-Side

Index.js

```
71     socket.on('join', function(params,callback) {
72         if (!isRealString(params.name) || !isRealString(params.room)) {
73             callback('Name and Room is required.');
74         }
75
76         socket.join(params.room);
77         //socket.leave('Basketball');
78
79         // io.emit() -> io.to('Basketball').emit()
80         // socket.broadcast.emit() -> socket.broadcast.to(..).emit()
81         //socket.emit() -> socket.to('Room name').emit()
82
83     socket.emit('newMessage',generateMessage('Admin','Welcome'));
84
        socket.broadcast.to(params.room).emit('newMessage',generateMessage('Ad
       min', `${params.name} has connected.`));
85
86     socket.on('createMessage',function(sender, msg, callback) {
87         if (msg.trim() != ''){
88             console.log(sender + ': ' + msg);
89             io.to(params.room).emit('newMessage',
               generateMessage(sender,msg));
90             callback(sender);
91         }
92     });
93
94     socket.on('createLocationMessage', function(coords){
95         io.to(params.room).emit('newLocationMessage',
           generateLocationMessage('Admin', coords.latitude,
           coords.longitude));
96 });
97
98         callback();
99     });
100 });
```

#76: Join a room that the user chooses.

#83: Send a ‘Welcome’ message to everybody in every room.

#84: Send a ‘Connected’ message to all room members except the one who connect.

#89: Send the messages to all the room members.

Client-Side

Index.html

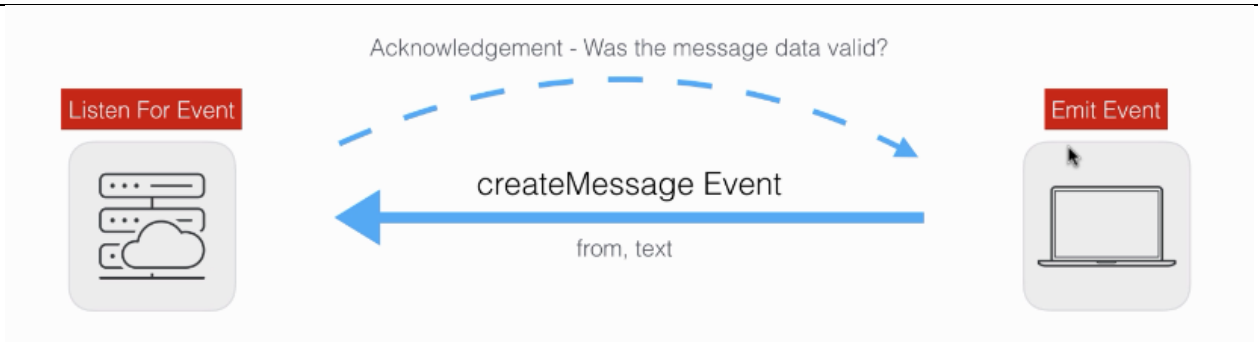
```
22 ▼    socket.on('connect', function(){
23          var params = jQuery.deparam(window.location.search);
24
25 ▼        socket.emit('join', params, function(err){
26 ▼            if (err) {
27                  alert(err);
28                  window.location.href = '/';
29 ▼            } else {
30                  console.log('No Error');
31              }
32          });
```

#23: Convert the URL query params to an Object of params.

#25: emit ‘join’ event when a user connect and send the params data with it.

#28: Redirect to the root route ‘join chat page’ when error happens.

Acknowledgment



It's a callback function that fires when the client (or the server) successfully receive the event.

We can pass the callback function as a last argument in `emit` and `on` function.

Client-Side

Index.html

```
5 ▼     $('form').submit(function(){
6 ▼         socket.emit('createMessage',from,
            $('.send-section input').val(),
            function(data){
7               console.log('The message from you
                ' + data + ' successfully
                received.');
8           });
```

#6: We Setup a callback function to invoke when the server received the event.

Server-Side

Index.js

```
64 ▼    socket.on('createMessage',function(sender, msg, callback) {
65          console.log(sender + ': ' + msg);
66          io.emit('newMessage', generateMessage(sender,msg));
67          callback(sender);
68      });
```

#64: We define the callback function that will be received from the client.

#67: We execute it.

Functions

socket.**on(**eventName**,** callbackFunction**);**

Catching event and response to it.

When 'event' happen **.on()** function catch it and fire a function as a response.

e.g. io.on('chat message', function(msg){});

socket.**emit(**eventName**,** eventValue**);**

socket.**send(**eventName**,** eventValue**);**

Emitting events (Sending data event {'server-side' from/to 'client-side'} to catch in the event

when the **user** types in a message, the **server** gets it as a `chat message` event.

- ✓ will emit to all the sockets connected to `/` the default namespace.

e.g. socket.emit('chat message', $('#m').val());

io.emit('chat message',msg);

io.emit('event', {key: 'value'});

Initialize a Port

var **io** = require('socket.io')**(3000);**

Will create an http server.

e.g. var io = require('socket.io')(80);

Note:

when I print something to the console in **server-side** 'index.js' they will be printed in the **CMD**.

And when we print something to the console in **client-side** 'index.html' they will be printed in the **browser console**.

Broadcast flag

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

add a `broadcast` flag to `emit` and `send` method calls.

Broadcasting means sending a message to everyone else except for the socket that starts it.

Note:

The difference between 'connection' and 'connect' events:

'**connection**' event : we write in the server-side to know **if a client connects to the server**.

'**connect**' event : we write in the client-side to know **if a server connects to the client**.
Note:
The difference between 'socket.emit()' and 'io.emit()' in the server-side: '**socket.emit()**': emit the event to single connection. '**io.emit()**' : emit the event to every connection.