WebSocket Or Socket.io

Socket.io is a library that allow you to implement real time communication between clients and the server without the need for users to restart the browser.

You can transform any data of kind between users like images, text videos and more.

REMEMBER = socket represent the client or user who is connected to the server.

Web socket is mainly compose of two parts:

* A server that integrates with the Node.JS HTTP Server [socket.io](https://github.com/socketio/socket.io) (meaning you create a node server and bind an instance of it to socket.io)
* A client library that loads on the browser side [socket.io-client](https://github.com/socketio/socket.io-client) (this give you access to all connected client object to the server and also other functions to invoke the serve)

During development, socket.io serves the client automatically for us, so there is no need for users to refresh their browser to receive data.

**To install socket.io (give you access to socket.io server functions)**

On your cli navigate to your project directory execute below command and then check your package.json to see if the depency is added.

npm install socket.io --save

**To create a socket.io connection**

Create a file index.js

Create an express server and store it in a variable. Then include the socket.io module in your project and bind the instance of your server to it.

const express = require('express');

const app = express();

//create express server and assign it to a variable

var server = app.listen(3000, () => {

  console.log('listening on \*:3000');

});

/\*include socket.io in your project and bind an instance of your server object to it  \*/

 const io = require("socket.io")(server);

Then define a route for your server homepage. (index.hml is a file in my project is not in any folder)

app.get('/', (req, res) => {

/\* "\_\_dirname" represent a source directory of a file on my computer.  so \_\_dirname will printe "C:\Users\justice ankomah\socketio"

  (Meaning the directory of this project on my computer. so \_\_dirname + "/index.html" will printe "C:\Users\justice ankomah\socketio/index.html")

     res.sendFile(file\_directory) means send a file to the user which takes in the directory of the file index.html on my computer \*/

  res.sendFile(\_\_dirname + "/index.html");

});

Now, socket.io is all about emiting and listening to event.

There are default event names created already by socket.io that you need to simple emit it or listen to it. You can also create your own event and listen to it.

“connection” = is a default event that will be fired by any client who connect to the server. So your job on the server is to listen to that event and have access to the object of the user who has connected to the server.

Use client\_object.on(“eventName”, callbackfunct) to create an event. Meaning use the connected client’s objects to create an event with a listener.

Then use the serve object io.emit(“client\_event\_name”, parameter) to emit or fire an event created by the client.

/\*

Below "connection" and "disconnect" event are defined by socket.io by defualt not custome one please\*

Listen to all new client side "connection" event and log to the console with

io.on("default\_io\_connection\_event", callback\_func\_with\_parameter\_of\_the\_client\_object);

 (check below index.html javascript side to see it for: var socket = io();

)

Create a connection event that will be fired by the client

(this function “io()” will fire this below event by default on the client whenever a new user connect to the server

The “socket” parameter will give you an object of the connected user); \*/

io.on("connection",(socket)=>{

 console.log("a user connnected"  );

 /\* Now the above "socket" parameter contains the connected client information. Use it to Check if user is disconnected with the "disconnect event"

 (this event will be fired authomatically when user refresh his browser or closese it)\*/

 socket.on("disconnect", ()=>{

   console.log("new user has just disconnected" );

 })

 /\* Create a cutome event and listener with callback\_func that contains a parameter to log what the user will send\*/

 socket.on('chat message', (msg) => {

    console.log(msg);

/\* fire the "chat message received" event created in the client index.html, javascript and pass the above parameter to it

(remember am using io.emit() not socket.emit() ) \*/

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

});

});

Then create and index.html file with embed inline javascript

<!DOCTYPE html>

<html>

  <head>

    <title>Socket.IO chat</title>

    <style>

      body { margin: 0; padding-bottom: 3rem; font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Helvetica, Arial, sans-serif; }

      #form { background: rgba(0, 0, 0, 0.15); padding: 0.25rem; position: fixed; bottom: 0; left: 0; right: 0; display: flex; height: 3rem; box-sizing: border-box; backdrop-filter: blur(10px); }

      #input { border: none; padding: 0 1rem; flex-grow: 1; border-radius: 2rem; margin: 0.25rem; }

      #input:focus { outline: none; }

      #form > button { background: #333; border: none; padding: 0 1rem; margin: 0.25rem; border-radius: 3px; outline: none; color: #fff; }

      #messages { list-style-type: none; margin: 0; padding: 0; }

      #messages > li { padding: 0.5rem 1rem; }

      #messages > li:nth-child(odd) { background: #efefef; }

    </style>

  </head>

  <body>

    <ul id="messages"></ul>

    <form id="form" action="">

      <input id="input" autocomplete="off" /><button>Send</button>

    </form>

   <!--EMBED SOCKET.IO-CLIENT JS FILE-->

   <!--LOAD WEBSOKET CDN CLIENT-->

   <script src="/socket.io/socket.io.js"></script>

   <script>

     //Fire the "connectiion" event in server

     var socket = io();

     var form = document.getElementById('form');

  var input = document.getElementById('input');

  /\* When a form event is trigered. Meaning when a form is submited.

  the "e" in the parameter will prevent the form from trying to send it to a backend server

  The preventDefault() method cancels the event if it is cancelable, meaning that the default action that belongs to the event will not occur.

For example, this can be useful when:

    Clicking on a "Submit" button, prevent it from submitting a form

    Clicking on a link, prevent the link from following the URL

  \*/

  form.addEventListener('submit', function(e) {

    e.preventDefault();

    //if the input form contains a value

    if (input.value) {

      // fire the event by sending the input values as a parameter

      socket.emit('chat message', input.value);

      //then set the inpute value to null

      input.value = '';

    }

  });

  // "create a chat event that is fired in the server and display the message"

  socket.on('chat message received', function(msg) {

    // create a <li> </li> elemtnet

    var item = document.createElement('li');

    // add the paramter sent from the emited event from the server as a value for the above created <li> </li>

    item.textContent = msg;

    // set the above <li> as child or inner element of the <ul> element in the html above

    messages.appendChild(item);

    //allow the window to scroll if the document height is longer the window

    window.scrollTo(0, document.body.scrollHeight);

  });

   </script>

  </body>

</html>

**Socket id**

Each client who connect to the server is assigned a specific id by socket.io by default.

When the user refresh his browser, he will be assign a new id (Because when he refresh his browser, the connection will be disconnected and reconnect back again as different user with new id)

These client id’s can be retrieve from both the client side and the server side for a particular user.

// server-side

/\*include socket.io and append a new instance of your server object to it  \*/

 const io = require("socket.io")(server);

io.on("connection", (socket) => {

  console.log(socket.id); // ojIckSD2jqNzOqIrAGzL

});

// client-side

 //Fire the "connectiion" event created in the server

     var socket = io();

// this event will be emited on the sever side automatically by socket.io when the client connect to the server

// So you just have to listen for this event and grab your id

socket.on("connect", () => {

  console.log(socket.id); // ojIckSD2jqNzOqIrAGzL

});

**Private messages to particular user id**

Because each user has it own particular id, you can send him a private message using his id.

//on the server

io.on("connection", socket => {

  /\*

   the client has to emit this event and pass in the id of another connected user and the message he want to send to the user

   then this event will be fired with the details he sent

  \*/

    socket.on("private message", (anotherSocketId, msg) => {

  /\* When the serve get the details above, he will fire it to that particular connected user id with your id (socket.id) and the message (msg)

    Then the client has to listen to this event and display it that “hey, john with id eh289geu has sent you this message”

  \*/

      socket.to(anotherSocketId).emit("private message", socket.id, msg);

    });

  });

**// in the Client**

/\*

 emit this event to the server with id of the connected user and the message you want to send to him

\*/

  socket.emit("private message", (anotherSocketId, msg));

  /\*

the sever will emit this event When someone send you a private message,

"socket. id" => below represent the id of the use who sent you the message

"msg" => represent the message the use has send to you

  \*/

  socket.on("private message",( socket.id, msg)=>{

   console.log("john with id " + socket.id +  " has sent you this message " + msg);

  });

**SOCKET USERNAME**

You can set a username to your socket by emiting below “set username” event on the client and pass in the socket name.

Also process the same event in the server and use the “socket.username” property to give your socket a name.

//in the client

  socket.emit("set username",("justice"));

 // in the server

   socket.on("set username", (username) => {

    socket.username = username;

  });

**To check if socket is connected to the server**

// ON THE CLIENT

//socket.connected will return boolearn (true or false)

  socket.on('connect', () => {

  console.log(socket.connected); // true

});

**To check if socket is disconnected to the server**

// ON THE CLIENT

socket.on('connect', () => {

  //socket.disconnected will return boolearn (true or false)

  console.log(socket.disconnected); // false

});

**ROOMS**

Room are different chat rooms a client or user can JOIN and LEAVE.

Please note that rooms are a **server-only** concept (i.e. the client does not have access to the list of rooms it has joined).

**To Join Room**

You can call socket.join(room\_Name) to subscribe the socket to a given channel:

// in the server

io.on('connection', socket => {

/\* you can emit a custome event from the client and handle it in the server here to

join a client to a room

\*/

    socket.join('some room Name');

  });

And then simply use the following to emit event from the sever and handle it in the client to send messages to groups.

// in the server

/\* create custome event that should be emited in the server and be handled

in the client to do the following\*/

/\* send message to all those in the given room including the sender \*/

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

0R

/\* send message to multiple rooms including the sender\*/

io.to('room1').to('room2').to('room3').emit('some event');

0R

io.on('connection', function(socket){

    //send message to all those in a room excluding the sender

    socket.to('some room name').emit('some event');

  });

**To leave a Room**

To leave a room simple use socket.leave(room\_Name); and follow the same approach as join.

And then redirect the user to home page after he clicks to leave a room.

## Disconnection

A user can lost internet connection or exit the browser without clicking on leave room button.

In this case you can’t use the “disconnect” event to track his id and the room names he was in. Beceause that event will execute after the user has

**To grab the socket id and the names of group the socket was in before he disconnect (refresh browser, lost internet connection, exit browser)**

**Use the “disconnecting” event and pass in socket.rooms() to get the user id and the names of rooms he was in before the disconnect.**

 io.on('connection', socket => {

    socket.on('disconnecting', () => {

// the Set contains at least the socket ID and (the names of the rooms he was in )

      console.log(socket.rooms);

    });

**EMIT CHEAT SHEET**

io.on("connection", (socket) => {

    // sending to a client  (eg. welcome john)

    socket.emit("hello", "can you hear me?");

    // sending to all clients except sender

    socket.broadcast.emit("broadcast", "hello friends!");

    // sending to all clients in "game" room except sender

    socket.to("game").emit("nice game", "let's play a game");

    // sending to all clients in "game1" and/or in "game2" room, except sender

    socket.to("game1").to("game2").emit("nice game", "let's play a game (too)");

    // sending to all clients in "game" room, including sender

    io.in("game").emit("big-announcement", "the game will start soon");

    // sending to all clients in namespace "myNamespace", including sender

    io.of("myNamespace").emit("bigger-announcement", "the tournament will start soon");

    // sending to a specific room in a specific namespace, including sender

    io.of("myNamespace").to("room").emit("event", "message");

    // sending to individual socketid (private message)

    io.to(socketId).emit("hey", "I just met you");

    // WARNING: `socket.to(socket.id).emit()` will NOT work, as it will send to everyone in the room

    // named `socket.id` but the sender. Please use the classic `socket.emit()` instead.

    // sending with acknowledgement

    socket.emit("question", "do you think so?", (answer) => {});

    // sending without compression

    socket.compress(false).emit("uncompressed", "that's rough");

    // sending a message that might be dropped if the client is not ready to receive messages

    socket.volatile.emit("maybe", "do you really need it?");

    // sending to all clients on this node (when using multiple nodes)

    io.local.emit("hi", "my lovely babies");

    // sending to all connected clients

    io.emit("an event sent to all connected clients");

  });

**Note:** The following events are reserved and should not be used as socket.emit() event names by your application:

* connect
* connect\_error
* disconnect
* disconnecting
* **newListener**
* **removeListener**
* // BAD, will throw an error
* socket.emit("disconnecting");