Send and Respond to Client > Server Messages

9 min

Good work! You've included a WebSocket server and client in the chat application and written code to confirm that they successfully connected. The next step is to start sending messages between them.

In this exercise, we're going to focus on how to send messages from a client to the server. The next exercise will focus on sending messages from server to client. As you complete this exercise and the next, take note of the similarities and differences between these processes.

The client-side WebSocket object has a .send() method that is used to send a message from the client to the server over the WebSocket connection between them. This .send() method accepts a single argument: the data to be sent to the server.

```
// Client
wsClient.send('This is the message that the server will receive.')
```

Note:

Preview: Docs Loading link description

Strings

can be sent using this .send() method however other

Preview: Docs Loading link description

data types

often need to be serialized first.

Now, let's have the server respond to this message. Remember the socket parameter that was passed to the callback defined for the 'connection' event? The value of this socket parameter is an object that represents each individual connection between a client and the server.

```
// Server
wsServer.on('connection', (socket) => {
  // What is this `socket` value ^ used for?
});
```

When the client sends a message to the server, a 'message' event is emitted and the socket connection object for the client that sent the message can listen for any messages coming into the server.

Like the WebSocketServer, the socket connection object is an instance of the EventEmitter class and also has the .on() method. To respond to these 'message'

Preview: Docs Loading link description

events

, we can pass in two arguments:

- The string 'message' for the event type
- A callback to define what the server should do when it receives a message. This callback receives the client's message as an argument, which we call data below:

```
// Server
// the `socket` argument represents the individual client connection
wsServer.on('connection', (socket) => {

  // the `socket` corresponding to the emitting client can detect 'message' events from that client
  socket.on('message', (data) => {
    console.log('message received: ' + data);
  })
})
```

In the example above, the .on() method is used on the socket parameter, which represents the WebSocket connection for the specific client that sent the message, and not on the WebSocketServer object (wsServer). The data argument passed to the callback holds the same data sent by the client which is printed to the console.

Instructions

Task 1

In **index.html**, take a look at the messageForm.onsubmit() handler in the DOM SETUP section. This event handler will be called each time the user submits a message. It displays the message the user typed in and then executes the function sendMessageToServer(message). As the name suggests, this function should take the message and send it along to the WebSocket connection to the server!

Find the sendMessageToServer() function in the WS CLIENT LOGIC section. You'll notice that we've started this function but have not yet sent the message to the server.

Below the comment labeled // Exercise 6, use the .send() method of the wsClient to send this message to the server.

Help

Your code should look something like this:

```
wsClient.send(message);
```

The wsClient variable should be initialized beforehand when the init() function is called and may be used to send data to the server. However, if something went wrong during initialization, the provided block of code will prematurely exit this function before attempting to send any data.

```
if (!wsClient) {
  showMessageReceived('No WebSocket connection :(');
  return;
}
```

Task 2

In server.js and within the callback defined for the 'connection' event, use the socket object's .on() method with the message event type to respond to client messages from that particular client. For now, console.log() the client's message.

Help

Remember, to call .on() on a socket, you need to have a socket object in scope. That means calling this function from inside the callback that runs when a server/client connection is initially established, like so:

```
wsClient.on('connection', (socket) => {
  socket.on('message', (data) => {
    // do something...
  });
});
```

Task 3

To test that your code works, restart your server, refresh the browser, and send a message from the client. You should see it logged to the console by your server!

Help

Press Control+c to stop the server and then enter the node server.js command again.