Send and Respond to Server > Client Messages

7 min

Great job! In the last exercise, you learned how to send messages from the client to the server, and how to make your server respond to those messages. This exercise will focus on the reverse: sending messages from the server to the client, and responding to those messages in the browser. These are very similar processes, so take note of the similarities as you complete this exercise.

To send a message from the server to a specific client, call .send() on the socket connection object; doing so will transmit the data passed to .send() along the specific client connection on which the method was called.

```
// Server
wsServer.on('connection', (socket) => {
  console.log('new connection!'); // this message is logged server side

  // this message is sent to the client when the client connects
  socket.send('Welcome to the server!');

   socket.on('message', (data) => {
    // the server will echo the message received back to the client
    socket.send('message received: ' + data);
  });
});
```

Again, .send() is called on the socket argument which represents the individual server/client connection. In the example above, a message is first sent to the client when the connection is made. Then, any time the server receives a message from a client, it will echo that message back to the same client.

Note: Like the client-side .send() method,

Preview: Docs Loading link description

Strings

may be sent using this method however other

Preview: Docs Loading link description

data types

may need to be serialized first.

To respond to server messages on the client side, we can assign a function to the.onmessage property of the client-side WebSocket object which will be called each time the

WebSocket client has received a message from the WebSocket server. Unlike the other event handlers which receive an Event object, .onmessage is called with a MessageEvent whose .data property holds the data sent by the server.

```
wsClient.onmessage = (messageEvent) => {
  const message = messageEvent.data;
  console.log('message received: ' + message);
};
```

In the example above, we first pull the message sent by the server from the .data property of the event parameter. Then it is printed to the console.

Instructions

Task 1

Let's start by echoing back the messages that clients send to the server.

In server.js, and within the callback defined for 'message' events, use the socket object's .send() method to send a message from a connected client whenever your server receives a message from that client. For now, send the same message received by the client.

Help

Remember, to execute an operation whenever your server receives a message from the client, place the code that performs that operation inside the callback passed to `.on('message', …)

```
// Server
wsServer.on('connection', (socket) => {
   socket.on('message', (data) => {
    // send a message to the socket here
  });
});
```

Task 2

In index.html, define the client-side WebSocket object's .onmessage event handler as a function with a messageEvent parameter. Then, get the message from the .data property of the event parameter and call showMessageReceived(), which we've defined for you, with the message received.

Help

For more information on how .onmessage works, consult the docs.

Task 3

Restart your server and then refresh your browser. Then, test that your code is working by typing in a message on the client and sending it. You should see it echoed back to you in the browser by the server!

Help

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