Respond to WebSocket Connections

9 min

Great work! You've created a WebSocket server and connected your browser client to that server. Next, you'll enable your server and client to respond whenever a connection is established between them.

Building a WebSocket application requires us to think in terms of

Preview: Docs Loading link description

events

and the code that should run when these events occur. Some events that occur in a WebSocket application might include:

- a *connection* is formed between client and server
- the server sends the client a *message* or vice versa
- the WebSocket connection is *closed*
- an *error* occurs in the WebSocket connection

Each of the events listed above has a name and a specific way to respond to that event. This exercise will focus on the 'connection' event, which is emitted each time a client completes the WebSocket handshake with a server, and on how to write server- and client- side code that responds to these connections.

First, let's explore responding to connections on the server-side with ws. As an instance of the EventEmitter class, a WebSocket.Server has the .on() method to define responses to events. To respond to 'connection' events, this method may be called with following arguments:

1. The string 'connection' as the event type
2. A callback to run each time a new 'connection' event is received by the server.

```
// Server
// wsServer is an instance of the ws WebSocket.Server class
wsServer.on('connection', (socket) => {
  // this code will run each time a new client connects to the server
  console.log('New client connected!');
})
```

The callback passed to the .on() method for the 'connection' event will receive a socket object as an argument. This object represents the specific WebSocket connection between the server and the client and is created for each unique client connected to the server. In the example above, we simply print out a message to the terminal verifying that a new client has connected. In the next exercise, we will begin to do a bit more with this socket value.

Note: *Check out the Node.js Essentials lesson from Learn Node.js for a refresher on the EventEmitter class.*

Instances of the browser's WebSocket class can also respond to these 'connection' events, but not with the .on() method. Instead, they have a property, .onopen that can be assigned a function that will be called when the client connects to a server.

```
// Client
// wsClient is an instance of the client-side WebSocket class
wsClient.onopen = (event) => {
  // this code will run when the client successfully connects to a server
  console.log('WebSocket connection established!');
}
```

In the example above, we print out a message to the console that a connection has been established. The function assigned to .onopen does receive an Event object as an argument, though this value isn't always used.

Instructions

Task 1

In server.js, find the comment labeled // Exercise 5 and define a response to the 'connection' event for your wsServer using the .on() method.

For now, the callback passed as the second argument can be an empty function.

Help

Pass 'connection' as the first argument to .on() and an empty callback as the second. Remember, the callback should accept a socket parameter. Your code should look like this:

```
wsServer.on('connection', (socket) => {
  // nothing here for now...
});
```

Task 2

In the callback that you pass to .on(), add console.log('new connection!') so that you can verify your connection is working.

You'll need to restart your server and refresh the browser to see this change reflected.

Help

Pass 'connection' as the first argument to .on() and an empty callback as the second. Remember, the callback should accept a socket parameter. Your code should look like this:

```
wsServer.on('connection', (socket) => {
  // nothing here for now...
});
```

Task 2

In the callback that you pass to .on(), add console.log('new connection!') so that you can verify your connection is working.

You'll need to restart your server and refresh the browser to see this change reflected.

Help

Your code may look something like this:

```
wsServer.on('connection', (socket) => {
  console.log('A new client has connected to the server!');
});
```

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

Task 3

In **index.html** and below the comment labeled // Exercise 5, define the WebSocket client's .onopen() event handler as a function that prints 'Connected to the WebSocket server!' to the browser console.

The event parameter will not be needed for this exercise.

Task 4

To test that your code works, restart your server and refresh the browser. You should see a message logged to the terminal window from which you ran your server as well as a message logged to the browser console.

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