Custom Message Types

13 min

Nice work! At this point you've implemented a functional chat room: users can send messages that are received by all other users. However, our application is still somewhat limited. Currently, our chat application only sends simple

Preview: Docs Loading link description

strings

between clients and the server. What if we wanted to send more complex data, like an object?

One situation in which using an object would be ideal is if we wanted to differentiate between different types of

Preview: Docs Loading link description

events

, such as when a new user joins vs. when a message is sent by a user.

A common pattern for accomplishing this is to use

Preview: Docs Loading link description

objects

with a .type property describing the type of message sent and an optional .payload property (typically also an object) containing any data to be transmitted:

```
// The message to be sent is included in the .payload object
{
  type: 'NEW_MESSAGE',
  payload: { message: 'Hello friend!!!' }
}

// A payload isn't always required though! Sometimes, simply distinguishing between message types can be useful
{
  type: 'NEW_USER'
}
```

Unfortunately, we can't simply send objects over a WebSocket connection since neither of the .send()

Preview: Docs Loading link description

methods

used by the client or the server accepts objects – but they do accept Strings!

All we need to do is convert the data we wish to send into a String, which we can easily do using the JSON.stringify() method.

```
// the sender stringifies the data before sending it
const message = {
  type: 'NEW_MESSAGE'
  payload: { message: 'Hello friend!' }
};
wsClient.send(JSON.stringify(message));
```

Above, the client wants to send a 'MESSAGE_RECIEVED' message with the actual message as a part of the .payload. Before sending the message, the object is stringified.

On the receiving end, we can use the JSON.parse() method which parses those strings and returns the objects that they represent.

```
// the receiver parses the data before interacting with it
const { type, payload } = JSON.parse(message);
switch (type) {
  case 'NEW_MESSAGE':
    // handle a new message received
    break;
  case 'NEW_USER':
    // handle a new user joining the server
    break;
  default:
    break;
}
```

In the example above, a

Preview: Docs Loading link description

switch

statement is used allowing the receiver to decide what to do with each type of message.

Note: When designing your own WebSocket applications, keep in mind that using a .type and a .payload property is not the only way to package your data. However, your sender and receiver must be aligned such that each side of the communication is using the same structure.

Instructions

Task 1

In the sendMessageToServer() function in **index.html**, find the comment labeled // Exercise 9 and modify the call to wsClient.send() such that it sends a custom message object with a .type and .payload property like the one below:

```
{
  type: 'NEW_MESSAGE',
  payload: { message: message }
}
```

Remember, the sender should use JSON.stringify() to serialize the message object.

Help

Your code should look something like this:

```
const data = {
  type: 'NEW_MESSAGE',
  payload: { message: message }
};
wsClient.send(JSON.serialize(data));
```

Task 2

In the wsClient.onopen() event handler, use wsClient.send() to send a custom message object to the server like the one below:

```
{
  type: 'NEW_USER'
}
```

Remember to use JSON.stringify() to serialize the message object.

Help

Not all custom message objects need a payload. In some cases, the .type alone can be enough information.

Task 3

Now, the client will send the server the custom message objects and the server will broadcast them out to all connected clients. When the client receives the message, it currently will show the message received as is. Before we can do this, we must parse the data and determine what to do with each type of message!

In the .onmessage handler, when the wsClient receives a new message, parse the custom message object into a string and extract the .type of the message and the .payload.

Help

Use JSON.parse() to convert the client's message from a string to an object. You may wish to further extract the type and payload into their own variables like so:

const { type, payload } = JSON.parse(customMessageObject);

Task 4

Still inside the .onmessage event handler, write a switch statement that switches on the message type. Create a different case for each type of message received.

Help

If you are following along with our examples, the switch statement will look like this:

```
switch (type) {
  case 'NEW_USER':
    break;
  case 'NEW_MESSAGE':
    break;
  default:
    break;
}
```

Task 5

Finally, call showReceivedMessage() with a different message format for each message type.

- The 'NEW_USER' type should simply display the text 'New user joined!'.
- The 'NEW_MESSAGE' type should display the message from the payload.

Help

Consider formatting incoming 'NEW_USER' messages in italics with the HTML element and 'NEW_MESSAGE' messages in bold with the HTML element to differentiate the two types of messages for the user.