

Topic	Video Chat App - Messaging	
Class Description	Students will learn how socket.io can be implemented in JavaScript, and complete the chat functionality of the application.	
Class	C-216	
Class time	45 mins	
Goal	<ul style="list-style-type: none"> Learning about socket.io and it's implementation in JavaScript Completing the chat functionality of our web-app! 	
Resources Required	<ul style="list-style-type: none"> Teacher Resources: <ul style="list-style-type: none"> Laptop with internet connectivity Earphones with mic Notebook and pen Visual Studio Code Student Resources: <ul style="list-style-type: none"> Laptop with internet connectivity Earphones with mic Notebook and pen Visual Studio Code 	
Class structure	Warm-Up Teacher - led Activity 1 Student - led Activity 1 Wrap-Up	10 mins 15 mins 15 mins 5 mins
WARM UP SESSION - 10mins		
Teacher Action		Student Action
<i>Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today?</i>		ESR: Hi, thanks, yes, I am excited about it!

Q&A Session	
Question	Answer
<p>Which function can be used to make the browser ask the user a question and save their response in a variable?</p> <p>A. prompt() B. log() C. ask() D. None of the above</p>	A
<p>Which event listener is used to check if the key was Pressed or not ?</p> <p>A. down() B. up() C. key() D. keydown()</p>	D
TEACHER-LED ACTIVITY - 15mins	
Teacher Initiates Screen Share	
<u>ACTIVITY</u> <ul style="list-style-type: none"> • Understanding the HTML and Bootstrap Code • Adding relevant HTML and CSS for responsiveness 	
Teacher Action	Student Action
<p>In the last class, we successfully set up the server for our video chat app. We learnt how NodeJS works, and how we can create backend servers with the help of it.</p> <p>We also implemented EJS, a view engine that ExpressJS</p>	

<p>uses!</p> <p>Do you have any doubts in the last class?</p> <p><i>Teacher clears the doubts, if any</i></p> <p>Great! Now we have already built the chat app with Python! Do you remember it?</p> <p>Awesome! Can you tell me what all we did to implement it back then?</p> <p>Awesome! Now, we have chat functionality in our video chat application as well, and we will work on it in this class with JS, to make it functional, before we finally work on the video chat functionality!</p> <p>Are you excited?</p> <p>Let's get started then!</p>	<p>ESR: Varied</p> <p>ESR: Yes</p> <p>ESR: We used the socket library and created a client and a server. We also made servers broadcast the messages received by the clients.</p> <p>ESR: Yes!</p>
<p>In the last class, if you recall, we installed socket.io into our project!</p> <p>It's time we finally use it now!</p> <p>Let's start with importing it in our server.js file</p> <p><i>Teacher opens the last class code in Visual Studio and writes the code in server.js</i></p>	

Note - The previous class code is available in [Teacher Activity 1](#) and [Student Activity 1](#)

```
const express = require("express");
const app = express();
const server = require("http").Server(app);
app.set("view engine", "ejs");
app.use(express.static("public"));

const { v4: uuidv4 } = require("uuid");

const io = require("socket.io")(server, {
  cors: {
    origin: '*'
  }
});
```

Here, we can see that we have created yet another constant with the name of **io**, in which we are requiring **socket.io** into our project.

After requiring it, we are also defining the server in which we want to use it, along with some options.

We had our server stored in the constant called **server**, therefore we are using that constant here, and in the options, we are setting the **cors' origin** to **"*"**.

Do you know what **cors** is?

In today's world, there are a lot of cyber security threats and errors, and along with how websites can be designed to avoid such threats, browsers are designed in a way to help avoid them too!

ESR:
Varied!

<p>Cors refers to cross origin resource sharing. When working with sockets, the sockets are sharing resources (or data) between client and server.</p> <p>Now, it may be the case that the application is designed in a way where the server is running on a different port, and the client is running in a different port.</p> <p>Browsers, these days, do not allow sharing the data from one port to another port. This means that if your client is running on a different port, and the server is running on a different port, then the browser will block the sharing of the data.</p> <p>To avoid such error, we set its origin to *, which means that we are letting the browser know to allow sharing data from all the ports, instead of blocking it.</p> <p>Now you may question that our client and server are running on the same port, so why set this up in the first place?</p> <p>The answer to that is simple! Sometimes, it could be the case that the server is listening to port 80, since we have an HTTP server, but the client might be using port 443, which is for HTTPS.</p> <p>In this case, the browser might block sharing the data, and to avoid it whatsoever, we set the origin for cors to *, to let the browser know not to block resource sharing in such cases.</p>	
<p>Similar to how we just imported socket.io in our server, we would also need to import it in our client, which is our HTML code!</p> <p>Let's add a script for that, in our index.ejs to import socket.io</p> <p><i>Teacher adds the following code in index.ejs</i></p>	

```

<!-- Bootstrap -->
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css">
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.5.1/jquery.min.js">
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js">
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js">

<!-- Socket.io -->
<script src="/socket.io/socket.io.js"></script>

```

We added the following line of code in our **<head>** tag, right below our Bootstrap files were added.

Next, in our **script.js** file, which contains our jQuery code, let's use the **socket.io** that we just imported in our HTML code to create a socket -

```

const socket = io("/");

$(function () {
  $("#show_chat").click(function () {
    $(".left-window").css("display", "none")
    $(".right-window").css("display", "block")
    $(".header_back").css("display", "block")
  })
})

```

Here, we have used **socket.io** with the **io("/")** function to create a **socket**, and saved it into a constant called **socket**.

This is the basic convention of how a socket can be created with the help of **socket.io** in client side HTML code.

Now let's pause for a bit and think about our flow!

<p>How will the users be sending messages?</p> <p>Good, so what would we want to do there?</p> <p>Excellent! And what will this listener do?</p> <p>Okay, and what will the server do?</p> <p>That's right! Finally, when the message is received back at the client, we can display the message!</p> <p>Now, we are clear on the steps! I will create the event listener that sends the message to the server, and then you will write the code for the server broadcasting the message and the client displaying the message!</p> <p>Now, the first thing that we need, before starting any of the coding, is to know the user's name.</p> <p>JavaScript has a unique function called <i>prompt()</i>, which can be used to make the browser ask the user a question and save their response in a variable.</p> <p>Let's quickly use it to know our user's name. We will make the changed in <i>script.js</i> -</p>	<p>ESR: By clicking on the send button.</p> <p>ESR: Add a click event listener.</p> <p>ESR: Send the message to the server.</p> <p>ESR: Broadcast the message to all the clients!</p>
<pre>const socket = io("/"); const user = prompt("Enter your name");</pre>	

With this, we will know our user's name! Next, we need to create an event listener for our send message button!

Let's take a look at the *index.ejs* and see our send button's code -

```
<div class="col-sm-12 col-md-12 col-lg-3 right-window">
  <div class="row" class="main-chat-window">
    <div class="col-sm-12 col-md-12 col-lg-12 messages" style="height: 81vh; background-color: #242f41;">
    </div>
    <div class="col-sm-12 col-md-12 col-lg-12" style="background-color: #242f41;">
      <div class="main_message_container">
        <input id="chat_message" type="text" autocomplete="off" placeholder="Type message here...">
        <div id="send" class="options_button">
          <i class="fa fa-plus" aria-hidden="true"></i>
        </div>
      </div>
    </div>
  </div>
</div>
```

Here, we can see that our send button has an *id* called **send**. We can also see that our *<input>* tag has an *id* called **chat_message**.

Let's use it to create our click event listener in *script.js* -

```
$(".header_back").click(function () {
  $(".left-window").css("display", "block")
  $(".right-window").css("display", "none")
  $(".header_back").css("display", "none")
})

$("#send").click(function () {
  if ($("#chat_message").val().length !== 0) {
    socket.emit("message", $("#chat_message").val());
    $("#chat_message").val() = "";
  }
})
```


Inside our **`$(function())`**, we have created another event listener on the **`send`** button. It is a **`click()`** event and inside this event, we are first checking if the message is empty or not, by checking the **`length`** of the message. We can fetch the value of the input field with the **`val()`** function.

If the message is not empty, we are using the **`socket.emit()`** function, where we are emitting a **`message`** with the value of our input field. This is the basic convention of sending data with **`socket.io`** in JavaScript.

Finally, we are making our input field's value to be empty, so the user can enter a new message if they want to!

Are we missing anything?

We handled the button click, but many of the times, the users just like to press the **`Enter`** key to send the message, right?

Let's create an event listener for that as well. We can use the **`keydown()`** event listener, and check if the key was **`Enter`** or not -

ESR:
Varied!

ESR:
Yes!

```
$("#send").click(function () {  
    if ($("#chat_message").val().length !== 0) {  
        socket.emit("message", $("#chat_message").val());  
        $("#chat_message").val() = "";  
    }  
})  
  
$("#chat_message").keydown(function(e){  
    if (e.key == "Enter" && $("#chat_message").val().length !== 0) {  
        socket.emit("message", $("#chat_message").val());  
        $("#chat_message").val() = "";  
    }  
})
```

Here, we have created a very similar event listener, but this time, we are listening for the **keydown** event, and we have this event listener on our **input** box. This is because our user might press enter right after typing the message, and they would still be in the input box.

Inside the function of the event listener, we are passing the event **e** to it, to know which key was pressed.

In the **if** condition of it, along with checking the length of the message to see if the message wasn't empty, we are checking for the key as well, if it was **Enter** or not.

If it was, and if the message wasn't empty, we are again emitting a **message** on the socket, and sending the message.

With this, half of our work is done! Now we just need to broadcast, or emit back this message to all the clients and

display the message in the chat box!	
Teacher Stops Screen Share	
STUDENT-LED ACTIVITY - 15 mins	
<ul style="list-style-type: none"> • Ask the student to press the ESC key to come back to the panel. • Guide the student to start Screen Share. • The teacher gets into Full Screen. 	
<p style="text-align: center;"><u>ACTIVITY</u></p> <ul style="list-style-type: none"> • Emit the message to all the clients • Display the message on the chat box 	
Teacher Action	Student Action
<i>Guide the student to get the boilerplate code from Student Activity 2</i>	<i>Student clones the code from Student Activity 2</i>
<p>Okay, now let's start by installing the modules in your project with the following command -</p> <p>yarn install</p> <p><i>Teacher helps the student in running the command</i></p>	<i>Student runs the command</i>
<p>Now, we have already emitted the message from our client, and we want to receive and emit it back to all the clients from the server.</p> <p>Let's do that! We will add it's code in server.js file</p> <p><i>Teacher guides the student in writing the code</i></p>	<i>Student writes the code</i>

```
app.get("/:room", (req, res) => {  
  res.render("index", { roomId: req.params.room });  
});  
  
io.on("connection", (socket) => {  
  socket.on("message", (message) => {  
    io.emit("createMessage", message);  
  });  
});
```

Here, you would notice that below our GET API, that we created in the last class, we are using **io.on("connection")**, which means that as soon as the socket establishes a connection with the client, it will shoot the arrow function next to it!

Now, inside the arrow function, we are passing the **socket** variable, which is the socket that we are using.

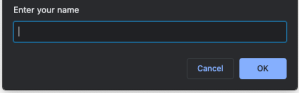

Inside the arrow function, we check for any **"message"** received, with the help of **socket.on("message")** function, and on receiving a message, the arrow function next to it shoots.

Inside this second arrow function, we are passing the **message** as a variable that we received from the client.

Finally, we are using the **io.emit()** function to trigger a **createMessage** socket event, and send the message we received by the client.

Now, this **createMessage** socket event is something that we will have to create on the client side, to render the message in the chat box!

<p>Let's write the code for it in our script.js now -</p> <p><i>Teacher guides the student in writing the code</i></p>	<p><i>Student writes the code</i></p>
 <pre> socket.on("createMessage", (message) => { \$(".messages").append(` <div class="message"> \${message} </div> `); }); </pre>	
<p>Here, below our \$ function, we are defining a socket event for createMessage, in which we are passing the message received from the server in the arrow function.</p> <p>Inside this function, we are taking the box with class messages (we already have this class implemented in the chat box), and appending some HTML to it.</p> <p>In this HTML, we are creating a <div> tag with class message, and inside this div tag, we are using a tag to display the message!</p> <p>With this! Our functionality seems to be completed!</p> <p>Let's test it now! Run the server with npm start command in the command prompt / terminal and open localhost:3030 in the browser</p> <p><i>Teacher helps the student in running the command and opening the server in the browser</i></p>	<p><i>Student runs the command and opens the server in the browser</i></p>

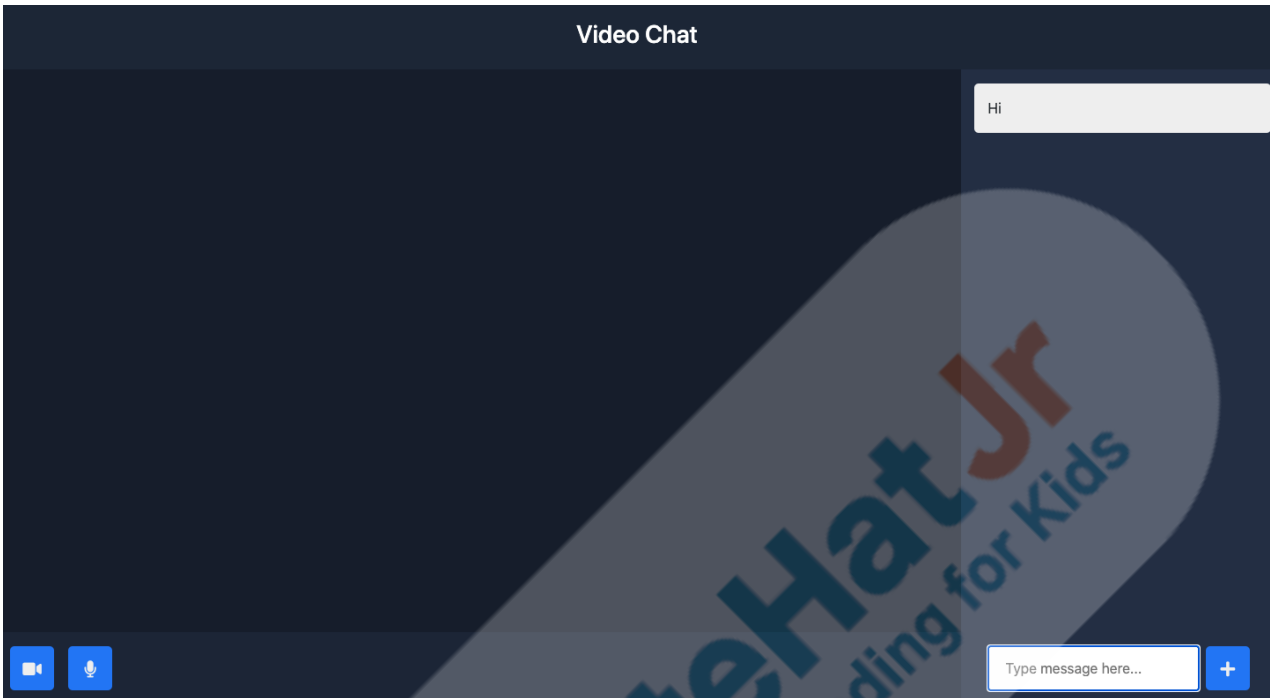
	
<p>The first thing you will notice is that it asks you to enter your name. This is because of the <i>prompt()</i> that we used.</p> <p>Enter your name and proceed!</p> <p>Next, type any message and see the output, if it is getting displayed on the browser?</p>	<p><i>Student enters their name</i></p> <p><i>Student enters a message and checks</i></p>
	
<p>As you can see, the message is being displayed, but it doesn't look good. Let's add some styling for it in <i>style.css</i></p> <p><i>Teacher guides the student in writing the code</i></p>	<p><i>Student writes the code</i></p>

```
49  .messages {  
50    display: flex;  
51    flex-direction: column;  
52  }  
53  
54  .message {  
55    display: flex;  
56    flex-direction: column;  
57  }  
58  
59  .message > span {  
60    background-color: #eeeeee;  
61    margin: 1rem 0;  
62    padding: 1rem;  
63    border-radius: 5px;  
64  }  
65
```

Now refresh the page and try again!

Student tries again

*Note - You might have to restart the server by running the **npm start** command again.*

 <p>The video chat interface shows a dark background with a large, faint watermark of the WhiteHat Jr logo. At the top, it says 'Video Chat'. On the right, there is a chat box with the text 'Hi'. At the bottom, there are icons for video and audio, and a text input field with the placeholder 'Type message here...' and a blue '+' button.</p>	
<p>Great! Do you remember NGROK? We can use it to test this application together.</p> <p>Let's try to run the ngrok server in a new command prompt / terminal</p> <p><i>Teacher guides the student to navigate to the directory where ngrok is, and run the command</i></p> <p>ngrok http 3030</p> <p><i>Note - Refer to Teacher Activity 3 / Student Activity 3 if it needs to be re-downloaded</i></p>	<p><i>Student runs the ngrok command</i></p>
<p><i>Refer to Teacher Activity 3 if it needs to be re-downloaded</i></p>	<p><i>Refer to Student Activity 3 if it needs to be re-downloaded</i></p>
<div style="background-color: black; color: green; padding: 10px; text-align: center; font-weight: bold;">ngrok http 3030</div>	

To see

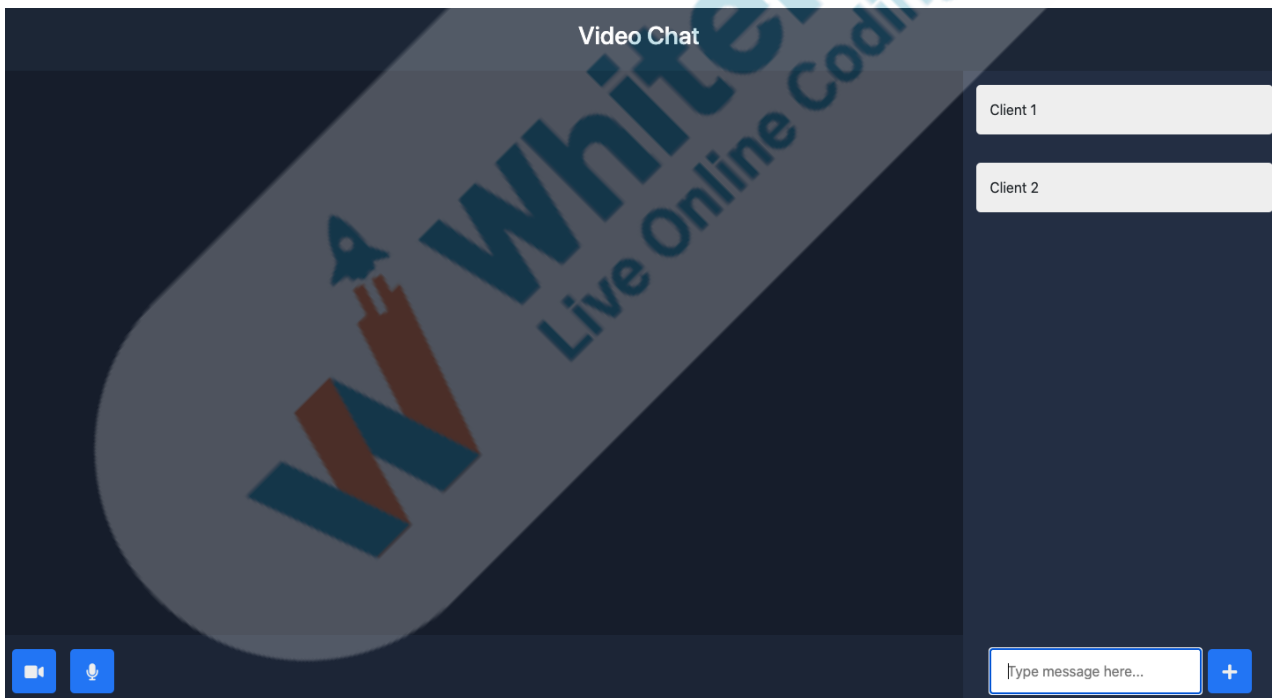
```
ngrok by @inconshreveable (Ctrl+C to quit)

Session Status      online
Session Expires     1 hour, 59 minutes
Version             2.3.40
Region              United States (us)
Web Interface        http://127.0.0.1:4040
Forwarding           http://07fe8b5b6a0f.ngrok.io -> http://localhost:3030
Forwarding           https://07fe8b5b6a0f.ngrok.io -> http://localhost:3030

Connections          ttl    opn    rt1    rt5    p50    p90
0                   0      0      0.00   0.00   0.00   0.00
```

Teacher copies the HTTPS URL, and pastes it in her browser.

Student and Teacher try to chat with each other.



Awesome! However, this still doesn't seem right.




We can see that our names are not getting displayed, which

© 2021 - WhiteHat Education Technology Private Limited.

Note: This document is the original copyright of WhiteHat Education Technology Private Limited.

Please don't share, download or copy this file without permission.

<p>is important.</p> <p>We can also notice that the Unique IDs of our URLs are different, meaning that we should be in different rooms, yet we can still chat with each other.</p> <p>In the next class, we will tackle these issues and learn about an awesome JavaScript library called PeerJS, the one that we installed at the beginning of this project!</p>	
Teacher Guides Student to Stop Screen Share	
WRAP UP SESSION - 5 Mins	
Quiz time - Click on in-class quiz	
Question	Answer
<p>What does Cors refer to?</p> <ul style="list-style-type: none"> A. catch origin resource sharing B. crane origin resource sharing C. cross origin resource sharing D. cost origin resource sharing 	C
<p>Is there any way to check if a "message" has been received?</p> <ul style="list-style-type: none"> A. msg.on("message") B. message.on("message") C. socket.on("message") D. socket.("message") 	C
<p>Which function can you use to emit a message with the value of the input field?</p> <ul style="list-style-type: none"> A. socket.e() B. s.emit() C. Socket.emit D. socket.emit() 	D

FEEDBACK <ul style="list-style-type: none"> Appreciate the students for their efforts in the class. Ask the student to make notes for the reflection journal along with the code they wrote in today's class. 	
Teacher Action	Student Action
<p>You get Hats off for your excellent work!</p> <p>In the next class</p>	<p><i>Make sure you have given at least 2 Hats Off during the class for:</i></p> <div> <div>Creatively Solved Activities  +10</div> <div>Great Question  +10</div> <div>Strong Concentration  +10</div> </div>
Project Discussion	
<div> <div>Teacher Clicks</div> <div>✕ End Class</div> </div>	
ADDITIONAL ACTIVITIES	
<p>Additional Activities</p> <p><i>Encourage the student to write reflection notes in their reflection journal using markdown.</i></p> <p>Use these as guiding questions:</p> <ul style="list-style-type: none"> What happened today? <ul style="list-style-type: none"> Describe what happened. The code I wrote. 	<p><i>The student uses the markdown editor to write her/his reflections in the reflection journal.</i></p>

- How did I feel after the class?
- What have I learned about programming and developing games?
- What aspects of the class helped me? What did I find difficult?

ACTIVITY LINKS		
Activity Name	Description	Link
Teacher Activity 1	Previous Class Code	https://github.com/pro-whitehatjr/PRO-C215-ReferenceCode
Teacher Activity 2	Reference Code	https://github.com/pro-whitehatjr/PRO-C216-ReferenceCode
Teacher Activity 3	NGROK	https://ngrok.com/download
Student Activity 1	Previous Class Code	https://github.com/pro-whitehatjr/PRO-C215-ReferenceCode
Student Activity 2	Boilerplate Code	https://github.com/pro-whitehatjr/PRO-C216-StudentBoilerplate
Student Activity 3	NGROK	https://ngrok.com/download