

Topic	Video Chat App - Streaming	
Class Description	Student will stream their video and audio across multiple peers with the help of PeerJS and WebRTC	
Class	C-220	
Class time	45 mins	
Goal	<ul style="list-style-type: none"> <li>Streaming video and audio across multiple peers</li> <li>Completing the video chat functionality</li> </ul>	
Resources Required	<ul style="list-style-type: none"> <li>Teacher Resources:               <ul style="list-style-type: none"> <li>Laptop with internet connectivity</li> <li>Earphones with mic</li> <li>Notebook and pen</li> <li>Visual Studio Code</li> </ul> </li> <li>Student Resources:               <ul style="list-style-type: none"> <li>Laptop with internet connectivity</li> <li>Earphones with mic</li> <li>Notebook and pen</li> <li>Visual Studio Code</li> </ul> </li> </ul>	
Class structure	<b>Warm-Up</b> <b>Student - led Activity 1</b> <b>Wrap-Up</b>	<b>10 mins</b> <b>30 mins</b> <b>5 mins</b>
<b>WARM UP SESSION - 10mins</b>		
<b>Teacher Action</b>		<b>Student Action</b>
<i>Hey &lt;student's name&gt;. How are you? It's great to see you!            Are you excited to learn something new today?</i>		<b>ESR:</b> Hi, thanks, yes, I am excited about it!

Q&A Session	
Question	Answer
How do you represent the closing of an HTML tag?  A. / B. // C. \ D. /-	<b>A</b>
Why should we use webRTC?  A. For real time communication B. For real time video C. For real time chat D. All of the above	<b>D</b>
STUDENT-LED ACTIVITY - 30mins	
Student Initiates Screen Share	
<b>ACTIVITY</b> <ul style="list-style-type: none"> <li>Understanding about WebRTC and it's functions</li> <li>Fetching the audio and the video for the chat app from the user's browser</li> </ul>	
Teacher Action	Student Action
<i>This is a student-led class where all activities should only be performed by the student on the repository that was updated on Heroku. The teacher is expected to guide the student on the code, explanations and steps.</i>	
In the last class, we stepped into the WebRTC domain and learned about it's advantages and usage. We also	

<p>displayed our own video on our video chat application.</p> <p>However, what if someone comes to the same room and wants to have a chat with you? We are not streaming our video currently, so our peers will not be streaming theirs either!</p> <p>In today's class, we will be learning how we can use the combination of sockets, PeerJS and WebRTC to stream video and audio data and complete our video chat functionality!</p> <p>Are you excited?</p> <p>Let's get started then!</p>	<p><b>ESR:</b> Yes</p>
<p>Up until now, we have used the <b>getUserMedia()</b> function to fetch the user's audio and video from the browser, and then added the stream into a <b>video</b> element that we created.</p> <p>What happens with this is that the users can see their own video. This means that even if 2 users are in the same room, though they can chat with each other through text chat, they can only see their own video.</p> <p>What we want to do here is to make these 2 users (or more than 2 users) see each other's videos as well.</p> <p>We discussed why we use WebRTC instead of sockets. Can you tell me why?</p>	<p><b>ESR:</b> Sockets work on client-server communication while WebRTC is peer to peer directly, without involving the server.</p> <p><b>ESR:</b></p>

<p>That's good! For peer to peer communication, what are we using?</p> <p>Awesome! So, do you think that with WebRTC already implemented, with the help of PeerJS, we can achieve the functionality of a video chat?</p>	<p>PeerJS</p> <p><b>ESR:</b> Yes!</p>
<p>Great! Then let's begin coding!</p> <p>Just like socket.js has socket events that can help broadcast and communicate data between server and client, PeerJS has peer events that can help broadcast and communicate data between peers!</p> <p>As soon as the user successfully joins the room in our server, we need to let the client know that the user is connected. This will help us trigger peer events that can help stream audio and video data to each other!</p> <p>We will start by broadcasting a socket event from the server to the client.</p> <p>Up until now, our socket handling in <b>server.js</b> looks like this -</p>	
<pre>io.on("connection", (socket) =&gt; {   socket.on("join-room", (roomId, userId, userName) =&gt; {     socket.join(roomId);     socket.on("message", (message) =&gt; {       io.to(roomId).emit("createMessage", message, userName);     });   }); });</pre>	

Now, we would need to broadcast a message to the client side, that the peer is connected to the server and is ready to stream their video!

Let's add the code for that in **server.js** -

*Teacher guides the student in writing the code*

*Student writes the code*

**Note** - All the coding must be done only in the directory/repository that is hosted on Heroku

```
io.on("connection", (socket) => {
  socket.on("join-room", (roomId, userId, userName) => {
    socket.join(roomId);
    io.to(roomId).emit("user-connected", userId);
    socket.on("message", (message) => {
      io.to(roomId).emit("createMessage", message, userName);
    });
  });
});
```

Here, we are emitting a message to the room, saying **user-connected** with the help of **io.to().emit()** function.

Also, we are sending the **userId** to the client to let it know which user we are talking about!

Next, we need to handle this socket event to trigger peer events, so let's do that right after we display the video stream in **script.js** -

*Teacher guides the student in writing the code*

*Student writes the code*

```
navigator.mediaDevices
  .getUserMedia({
    audio: true,
    video: true,
  })
  .then((stream) => {
    myStream = stream;
    addVideoStream(myVideo, stream);

    socket.on("user-connected", (userId) => {
      connectToNewUser(userId, stream);
    });
  })
```

Here, we are using the **socket.on()** function to handle the **user-connected** event, and we are calling a function called **connectToNewUser()** with 2 arguments -

1. **userId** - To uniquely identify the user
2. **stream** - Our audio and video stream, which we want to display to others

Now, we need to send this stream to other users, so that they can, too, view our video and listen to our audio!

For that, PeerJS has a special event called **call**, which is used to call fellow peers to stream data, or answer their calls to receive their streams!

Let's use this function to share our stream with other users!

*Teacher helps the student in writing the code*

*Student opens the file*

```
function connectToNewUser(userId, stream) {  
  const call = peer.call(userId, stream);  
  const video = document.createElement("video");  
  call.on("stream", (userVideoStream) => {  
    addVideoStream(video, userVideoStream);  
  });  
};
```

Here, we have the function **connectToNewUser()** which receives the **userId** and **stream** of the user who just got into the room.

Inside the function, we first make a call to the peers with a **peer.call()** function, giving our peers our uniqueId and stream. We are also storing this call in a constant called **call**

Now we have 2 scenarios when we join a new room -

1. There are already people waiting in the room to video chat
2. There is someone new coming into the room

Both of these scenarios are applicable. When we call **connectToNewUser()** function, we are actually new to the room, and while we are making calls to all who are already in the room, we also want to start accepting their streams!

For this, we are creating a **video** element in this function, just like how we did it earlier in which we streamed our video, we will use this video element to stream their video.

Next, we are using our **call** constant to handle the **stream** event on it with the **call.on()** function. **stream** is the event that gets triggered and gives you all the streams.

<p>In this case, it would be the streams of the existing users already in the room.</p> <p>Inside this <b>stream</b> event, we are calling the <b>addVideoStream()</b> function we created in the last class, to add their video streams and display it to us!</p>	
<p>Next, we need to handle the second scenario, which is, to accept the stream from a new user who has just joined the room!</p> <p>For this, just as a new user uses the <b>peer.call()</b> function to call the peers, we will need to <b>answer</b> any incoming calls.</p> <p>Also, we need to create this event function on answering inside the <b>then()</b> function of our stream, so that this function is valid as long as we are streaming.</p> <p>Let's do that -</p> <p><i>Teacher helps the student in writing the code</i></p>	<p><i>Student writes the code</i></p>



```
navigator.mediaDevices
  .getUserMedia({
    audio: true,
    video: true,
  })
  .then((stream) => {
    myStream = stream;
    addVideoStream(myVideo, stream);

    socket.on("user-connected", (userId) => {
      connectToNewUser(userId, stream);
    });

    peer.on("call", (call) => {
      call.answer(stream);
      const video = document.createElement("video");
      call.on("stream", (userVideoStream) => {
        addVideoStream(video, userVideoStream);
      });
    });
  })
})
```

Here, we are handling the peer event of the **call** with the help of the **peer.on()** function. In this function, we use the **call.answer()** function, to answer the call with our **stream**.

Notice that earlier, when we called **call.on()** event on stream, we were receiving this stream that our peers answer with to our calls.

Once we answer their call with our stream, we also need to

display their stream in our client, so again we create a **video** element, and again we use the **call.on()** event handler for **stream** events and use the **addVideoStream()** function to display their streams!

To sum this up, in our

1. first scenario where there were already users in the room, as soon as we connect, we make a call to all the peers with our stream. We then wait for their streams as an answer, and start displaying their streams as they appear.
2. In the second scenario, when someone new joins, we are answering their call with our stream, and displaying their stream in our call.

Think of this as 2 scenarios that would always be there and we are handling both of them.

Now, we are all done with the video streaming! Let's test our code.

For that, we will need to push our code on Heroku.

Do you remember how we do it?

**ESR:**

```
git add -A
git commit "video chat
done"
git push
And then "deploy branch"
from the dashboard in
Heroku!
```

Awesome! Then let's run those commands in your terminal/command prompt, deploy it and check the output!

<p><b>Manual deploy</b></p> <p>Deploy the current state of a branch to this app.</p>	<p>Deploy a GitHub branch</p> <p>This will deploy the current state of the branch you specify below. <a href="#">Learn more.</a></p> <p>Choose a branch to deploy</p> <div> <input type="text" value="main"/> <span>⌵</span> </div> <div>Deploy Branch</div>
<p>Now let's open the app and test the code!</p> <p><i>Student and teacher opens the same link, enters the same room and tries to video chat with each other</i></p>	<p><i>Student and teacher opens the same link, enters the same room and tries to video chat with each other</i></p>
<p>Awesome! We have finally done it.</p> <p>Congratulations on building this Video Chat App!</p> <p>However, we are still now done yet! We have 2 buttons, to stop video and mute audio, and we still have not handled their click events!</p> <p>Let's quickly do that!</p> <p><i>Teacher guides the student in writing the code</i></p>	<p><i>Student writes the code</i></p>

```
<div class="col-sm-12 col-md-12 col-lg-12 options">
  <!-- Icons -->
  <div id="stop_video" class="options_button">
    <i class="fa fa-video-camera"></i>
  </div>
  <div id="mute_button" class="options_button">
    <i class="fa fa-microphone"></i>
  </div>
  <div id="show_chat" class="options_button">
    <i class="fa fa-comment"></i>
  </div>
</div>
```

If you take a look, our mute button has a id ***mute\_button*** and our stop video button has id ***stop\_video***

Using these id, we can create click events for these buttons in our ***\$(function(){})***.

Let's do that -

*Teacher helps the student in writing the code*

*Student writes the code*

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

Let's focus on the mute button first! For the mute button, we need to check if our stream is accepting any audio or not, and simply toggle it.

This means that if our stream accepts audio, then we need to stop it and if it's not, then we need to allow it!

To check if our stream is accepting audio or not, we can use a function called **getAudioTracks()** on our stream!

Let's do that -

*Teacher guides the student in writing the code*

*Student follows the instructions*

```
$("#mute_button").click(function () {  
    const enabled = myStream.getAudioTracks()[0].enabled;  
    if (enabled) {  
        myStream.getAudioTracks()[0].enabled = false;  
    } else {  
        myStream.getAudioTracks()[0].enabled = true;  
    }  
})
```

Here, we are using the function to check if the audio is enabled or not, and toggling it accordingly with the help of if-else statements.

Next, we also need to change the icons. If the audio is not there, then we need to add a slash on the mic!

Let's create HTML for that!

*Teacher helps the student in writing the code*

*Student writes the code*

```
$("#mute_button").click(function () {  
    const enabled = myStream.getAudioTracks()[0].enabled;  
    if (enabled) {  
        myStream.getAudioTracks()[0].enabled = false;  
        html = `<i class="fas fa-microphone-slash"></i>`;   
    } else {  
        myStream.getAudioTracks()[0].enabled = true;  
        html = `<i class="fas fa-microphone"></i>`;   
    }  
})
```

Lastly, to make it more appealing, we can toggle the colors of the icon, blue if it is enabled and red if it disabled!

For that, let's add a class and some styling for that class!

*Teacher helps the student in writing the code*

*Student writes the code*

```
$("#mute_button").click(function () {
  const enabled = myStream.getAudioTracks()[0].enabled;
  if (enabled) {
    myStream.getAudioTracks()[0].enabled = false;
    html = `<i class="fas fa-microphone-slash"></i>`;
    $("#mute_button").toggleClass("background_red");
  } else {
    myStream.getAudioTracks()[0].enabled = true;
    html = `<i class="fas fa-microphone"></i>`;
    $("#mute_button").toggleClass("background_red");
  }
})
```

And it's styling in **style.css** would be -

```
.background_red {
  background-color: #f6484a;
}
```

Finally, we need to change the HTML inside the button, to change the icon -

*Teacher helps the student in writing the code*

*Student writes the code*



```
$("#mute_button").click(function () {
    const enabled = myStream.getAudioTracks()[0].enabled;
    if (enabled) {
        myStream.getAudioTracks()[0].enabled = false;
        html = `<i class="fas fa-microphone-slash"></i>`;
        $("#mute_button").toggleClass("background_red");
        $("#mute_button").html(html)
    } else {
        myStream.getAudioTracks()[0].enabled = true;
        html = `<i class="fas fa-microphone"></i>`;
        $("#mute_button").toggleClass("background_red");
        $("#mute_button").html(html)
    }
})
```

The same code also goes in **stop\_video** click event handler, but instead of using **getAudioTracks()**, we will be using **stopVideoTracks()** and also, the icon's HTML would change, replacing **microphone** with **video** -

*Teacher helps the student in writing the code*

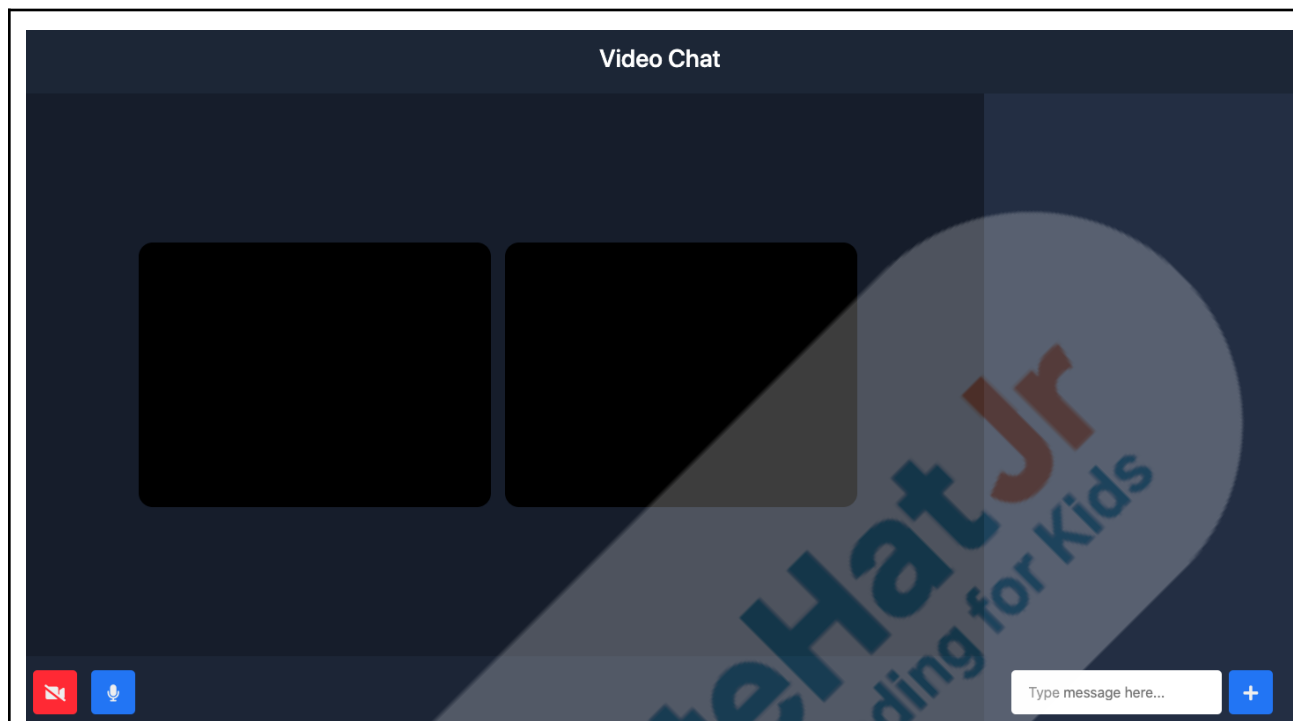
*Student writes the code*

```
$("#stop_video").click(function () {  
    const enabled = myStream.getVideoTracks()[0].enabled;  
    if (enabled) {  
        myStream.getVideoTracks()[0].enabled = false;  
        html = `<i class="fas fa-video-slash"></i>`;   
        $("#stop_video").toggleClass("background_red");  
        $("#stop_video").html(html)  
    } else {  
        myStream.getVideoTracks()[0].enabled = true;  
        html = `<i class="fas fa-video"></i>`;   
        $("#stop_video").toggleClass("background_red");  
        $("#stop_video").html(html)  
    }  
})
```

Let's push it on Heroku and check if the buttons work fine!

*Teacher helps the student in pushing the code to Heroku and test the output*

*Student pushes the code to Heroku and test the output*






Awesome! Our functionality is complete! In the next class, we will be looking into inviting people into our video chat app!

**Teacher Guides Student to Stop Screen Share**

**WRAP UP SESSION - 5 Mins**

**Quiz time - Click on in-class quiz**

Question	Answer
<p>What are the key features of Node.js?</p> <ul style="list-style-type: none"> <li>A. servers for Web Applications</li> <li>B. Real time data</li> <li>C. Used for network applications</li> <li>D. All of the above</li> </ul>	<b>D</b>

<p>Which of the modules is used with node.js to create web pages?</p> <p>A. http B. Smtip C. Url D. connect</p>	<p><b>A</b></p>
<p>Which tag tells the browser where the page starts and stops?</p> <p>A. Head B. HTML C. Body D. None of the above</p>	<p><b>A</b></p>
<p><b>End the quiz panel</b></p>	
<p><b><u>FEEDBACK</u></b></p> <ul style="list-style-type: none"> <li>• <b>Appreciate the students for their efforts in the class.</b></li> <li>• <b>Ask the student to make notes for the reflection journal along with the code they wrote in today's class.</b></li> </ul>	
Teacher Action	Student Action
<p>You get Hats off for your excellent work!</p> <p>In the next class, we will be building an emailer to send invites to our friends and family through email!</p>	<p><i>Make sure you have given at least 2 Hats Off during the class for:</i></p> <div data-bbox="1036 1417 1323 1522"> <p>Creatively Solved Activities  +10</p> </div> <div data-bbox="1036 1543 1323 1648"> <p>Great Question  +10</p> </div> <div data-bbox="1036 1669 1323 1774"> <p>Strong Concentration  +10</p> </div>

<h2>Project Discussion</h2> <p>In Class 220, we learnt how with the help of PeerJS, we can stream our video and audio to other peers. In this project, you'll work on handling PeerJS call events in a voice call application.</p> <p>Alison, your friend, has been working on a personal project to create a voice call app. She has completed most of the work, but is still struggling with accepting incoming calls from her peers. Help her complete the code so that people can use the voice chat application.</p>	
<div>Teacher Clicks</div> <div>✕ End Class</div>	
<h3>ADDITIONAL ACTIVITIES</h3>	
<h4>Additional Activities</h4> <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"> <li>• What happened today?       <ul style="list-style-type: none"> <li>◦ Describe what happened.</li> <li>◦ The code I wrote.</li> </ul> </li> <li>• How did I feel after the class?</li> <li>• What have I learned about programming and developing games?</li> <li>• What aspects of the class helped me? What did I find difficult?</li> </ul>	<p><i>The student uses the markdown editor to write her/his reflections in the reflection journal.</i></p>

ACTIVITY LINKS		
Activity Name	Description	Link
Teacher Activity 1	Previous Class Code	<a href="https://github.com/pro-whitehatjr/PRO-C219-ReferenceCode">https://github.com/pro-whitehatjr/PRO-C219-ReferenceCode</a>
Teacher Activity 2	Reference Code	<a href="https://github.com/pro-whitehatjr/PRO-C220-ReferenceCode">https://github.com/pro-whitehatjr/PRO-C220-ReferenceCode</a>
Student Activity 1	Previous Class Code	<a href="https://github.com/pro-whitehatjr/PRO-C219-ReferenceCode">https://github.com/pro-whitehatjr/PRO-C219-ReferenceCode</a>