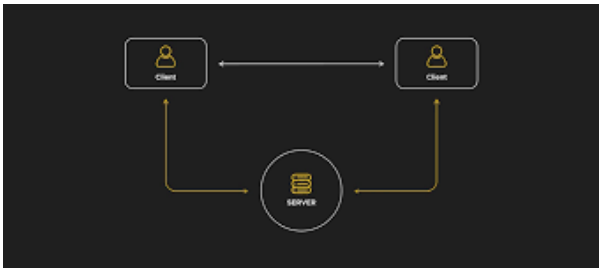
VIDEO CHAT APP - Streaming



What is our GOAL for this CLASS?

In this class, we have streamed the video and audio across multiple peers with the help of PeerJS and WebRTC.

What did we ACHIEVE in the class TODAY?

- Streaming video and audio across multiple peers
- Completing the video chat functionality

Which CONCEPTS/ CODING BLOCKS did we cover today?

- To learn how we can use the combination of sockets, PeerJS and WebRTC to stream video and audio data.

How did we DO the activities?

In the last class, we stepped into the WebRTC domain and learned about it's advantages and usage. We also displayed our own video on our video chat application.

Activity:

1. Our socket handling in ***server.js*** :

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

2. Now, 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 in ***server.js.***

```
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);
        });
    });
});
```

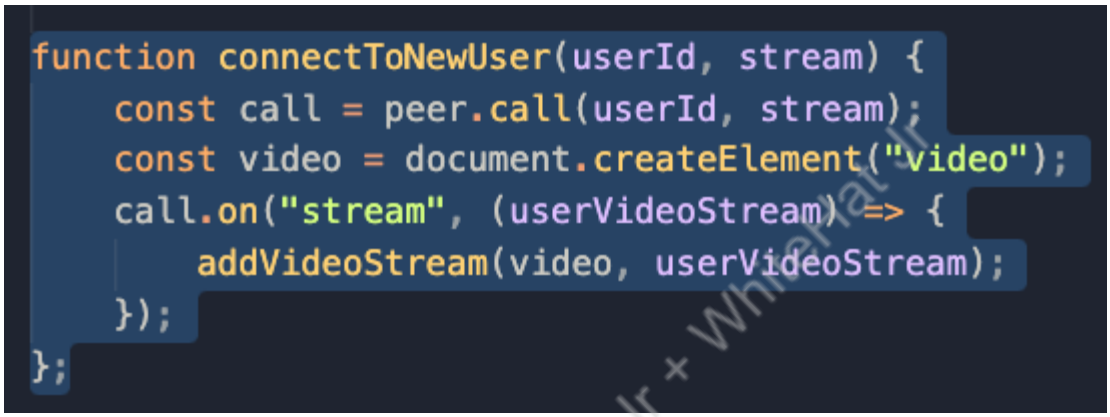
3. We need to handle the socket event to trigger peer events, so let's do that right after we display the video stream in ***script.js***.
   The ***socket.on()*** function handle the ***user-connected*** event, and we are calling a function called ***connectToNewUser()*** with 2 arguments -

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

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

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

4. Now, send this stream to other users, to let them view our video and listen to our audio. Function ***connectToNewUser()*** which receives the ***userId*** and ***stream*** of the user who just got into the room. Inside the function, we will 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 "***call".***



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

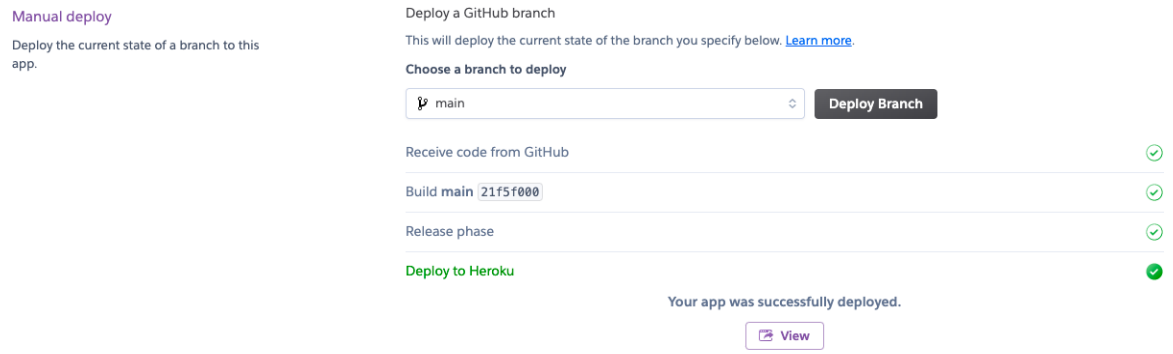
5. Next, we need to handle the second scenario, which is, to accept the stream from a new user who has just joined the room. To handle the peer event of the ***call*** with the help of the ***peer.on()*** function. In this function, we use the ***call.answer()*** function, to answer the call with our ***stream***. Create a ***video*** element, and use the ***call.on()*** event handler for ***stream*** events and use the ***addVideoStream()*** function to display their streams.

```
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);
            });
        });
    })
```


6. Let's test our code.
   For that, we will need to push our code on github and then deploy it on Heroku.
   git add -A
   git commit "video chat done"
   git push



7. We need to add the events to handle the click event of video and mute audio.

```
<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>
```

8. We can create click events for these buttons in our ***$(function(){})***.

```
$("#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(){

})
```


9. For the mute button, we need to check if our stream is accepting any audio or not, and simply toggle it. We can use a function called ***getAudioTracks()*** on our stream.

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

10. 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.

```
$("#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>`;
    }
})
```

11. To make it more appealing, we can toggle the colors of the icon, blue if it is enabled and red if it disabled.

```
$("#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;
}
```

12. Finally, we need to change the HTML inside the button, to change the icon.

```
$("#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)
    }
})
```


13. The same code also goes in ***stop_video*** click event handler, but we will be using ***stopVideoTracks()*** and also, the icon's HTML would change, replacing ***microphone*** with ***video***.

```
$("#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)
    }
})
```

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

What's NEXT?

In the next class, we will complete our video chat functionality for this application.

Expand Your Knowledge:

Explore more about streaming here.