

| Topic   | Video Chat App - WebRTC  |   |
|---|--|---|
| Class Description   | Student will learn about WebRTC and how Video and Audio can be gathered from the user on the browser and stream it to their fellow peers.  |   |
| Class   | C-219  |   |
| Class time  | 45 mins  |   |
| Goal  | <ul style="list-style-type: none"> <li>• Learning about WebRTC</li> <li>• Fetching Audio and Video of the users from the browser</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><br><b>Student - led Activity 1</b><br><b>Wrap-Up</b>  | <b>10 mins</b><br><b>30 mins</b><br><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!<br/>           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         |
| What is jQuery?<br><br>A. Jargon library<br>B. Java query<br>C. Javascript library<br>D. None of the above  | <b>C</b>       |
| What does HTML stand for?<br><br>A. Hyper Text Markup Language<br>B. Hyper Test Markup language<br>C. High Text Markup language<br>D. None of the above   | <b>A</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 deployed our application on Heroku and saw how it can be hosted online and accessed by  |                |

|  |  |
|--|--|
| <p>anyone in the world without you having to run it locally on your device!</p> <p>We also learned that this process is known as DevOps!</p> <p>Now, you must be wondering, how did it happen automatically, as soon as the code is pushed on Heroku while you have to do so much work locally, right from installing the packages to running the server.</p> <p>Can you guess?</p> <p>The answer to that is the <b>package.json</b> file. As soon as the code is pushed into Heroku, heroku is smart enough to know which language the code is written in because of it!</p> <p>Accordingly, since it knows that the application is created using NodeJS, it installs the NodeJS file and runs the mandatory <b>yarn install</b> command to install the node command.</p> <p>Once done, it automatically runs the <b>npm start</b> command and the application is up and running on Heroku!</p> <p>Great, isn't it?</p> <p>Now, we are almost done with our video chat app! One last bit of main functionality that remains is the actual streaming of Audio and Video from peer to peer and in today's class, we are going to do just that!</p> <p>Are you excited?</p> <p>Let's get started then!</p> | <p><b>ESR:</b><br/>Varied</p> <p><b>ESR:</b><br/>Yes!</p> <p><b>ESR:</b><br/>Yes</p> |
|--|--|

Earlier, in the previous classes, we discussed how even big video chat apps like Google Meet use a combination of WebRTC and PeerJS in their applications!

By now, we already know what PeerJS is, and we have also implemented it in our code to make our chat functionality work with rooms!

Now, we will be learning about WebRTC to understand how browsers offer some APIs and functions to get user's video and audio and stream it to others.

Do you remember the full form of WebRTC?

**ESR:**  
Web Real Time Chat

That's great! As the name itself suggests, WebRTC is an open source project for browsers (and even mobile applications) to provide them with real time chat functionalities with the help of simple APIs.

Now, you may wonder, why not simply use sockets, or what does WebRTC do differently than sockets?

The answer is simple, both of them do the same thing, but differently.

**ESR:**  
Yes!

What this means is that sockets are used for client-server communication. When we implemented our chat functionality, we used sockets where we were sending the message from client to the server, and then broadcasting those messages to all the clients.

**ESR:**  
Yes!

Imagine doing the same thing, but with Videos and Audios. Videos and Audios would be constantly streaming from all the clients, with rates of 30 to 60 frames per second.

Now, imagine maintaining all the data of audio and video for all the frames of all the seconds for all the clients on the server while also broadcasting it at the same time. You

|   |                            |
|---|----------------------------|
| <p>would also need to ensure that the data is not getting delayed or the experience of the users will be hindered. Do you think it's feasible?</p> <p>That's where WebRTC comes into play! It simply sends data from peer to peer (or client to client) without the server's interference, and works best with PeerJS!</p> <p>Now with the scenarios that we have just imagined, since we have to continuously receive and send audio and video streams back and forth, WebRTC is the right choice.</p> | <p><b>ESR:</b><br/>No!</p> |
| <p>Let's begin coding!</p> <p>The first thing we would want to do is to get organised as to what needs to be done.</p> <p>In our output so far, we have a box on the left side of the screen, potentially where we want to display the videos of the users -</p>  |                            |

|  |                                       |
|--|---------------------------------------|
|   |                                       |
| <p>Let's take a look at the HTML code for our application so far!</p> <p><i>Teacher guides the student to open the application in VS Code and open index.ejs file</i></p> <p><b>Note - All activities must be performed only in student's device, on the same directory that was pushed to heroku in the last class as the same code will be updated on Heroku</b></p> |                                       |
| <pre>&lt;div class="row main"&gt;   &lt;div class="col-sm-12 col-md-12 col-lg-9 left-window"&gt;     &lt;div class="row"&gt;       &lt;div class="col-sm-12 col-md-12 col-lg-12" style="height: 81vh; background-color: #171e2a;"&gt;         &lt;!-- Video Streams --&gt;       &lt;/div&gt;     &lt;/div&gt;   &lt;/div&gt; &lt;/div&gt;</pre>                       |                                       |
| <p>Here, we can notice that there is a div container in which we will display the video streams. Let's give this <b>div</b> and <b>id</b> called <b>video_grid</b> -</p>   | <p><i>Student writes the code</i></p> |

```
<div class="col-sm-12 col-md-12 col-lg-12" style="height: 81vh; background-color: #171e2a;"
  id="video_grid">
  <!-- Video Streams -->
</div>
```

Great! Now, we will be able to identify the div we want to add the streams to in our script!

Now comes the interesting part, where we will write the code for the browser to take our video and audio!

Let's start by opening the **script.js** file -

Student opens the file

```
const user = prompt("Enter your name");

const myVideo = document.createElement("video");
myVideo.muted = true;
```

Here, after asking the user to enter their name as a **prompt**, we create a new **video** element. It's just like any other tags that we have come across so far, such as **<div>**, **<span>**, **<a>**, **<p>**, **etc.** and it is used to display a video.

We are creating this **video** element with the help of JavaScript function called **createElement()**, and we are creating it in the **document**, which refers to all the contents of the page.

We are also saving this **video** element we just created in a variable called **myVideo**, which we can use later!

This video tag will be displaying our video.

Now, we don't want our video to speak on our device, so we will have to mute our video. We can do this by setting the **muted** attribute of the video we just created to **true**, which will mute our video on our device.

The element we just created is to display our video, but we will also have a stream that will be displayed.

Let's create a separate variable for that -

```
const user = prompt("Enter your name");

const myVideo = document.createElement("video");
myVideo.muted = true;

let myStream;
```

This **myStream** variable will contain our Audio and Video stream that we will display in **myVideo**.

We have the variables set, and hence we are all set to fetch our video and audio stream!

For that, there is a special function in JavaScript, known as **getUserMedia()**.

The way it is used is **navigator.mediaDevices.getUserMedia()**

Let's understand this better. Open the google inspect, and type **navigator** in the console -



```
> navigator
< Navigator {vendorSub: "", productSub: "20030107", vendor: "Google Inc.", maxTouchPoints: 0, getUserMedia: f, ...}
  appName: "Netscape"
  appVersion: "5.0 (Macintosh; Intel Mac OS X 10_15_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36"
  > bluetooth: Bluetooth {}
  > clipboard: Clipboard {}
  > connection: NetworkInformation {onChange: null, effectiveType: "4g", rtt: 50, downlink: 10, saveData: false}
  cookieEnabled: true
  > credentials: CredentialsContainer {}
  deviceMemory: 0
  doNotTrack: null
  > geolocation: Geolocation {}
  > getUserMedia: f ()
  hardwareConcurrency: 4
  > hid: HID {onconnect: null, ondisconnect: null}
  > keyboard: Keyboard {}
  language: "en-GB"
  > languages: (3) ["en-GB", "en-US", "en"]
  > locks: LockManager {}
  > managed: NavigatorManagedData {onmanagedconfigurationchange: null}
  maxTouchPoints: 0
  > mediaCapabilities: MediaCapabilities {}
  > mediaDevices: MediaDevices {ondevicechange: null, getUserMedia: f}
  > mediaSession: MediaSession {metadata: null, playbackState: "none"}
  > mimeTypes: MimeTypeArray (0: MimeType, 1: MimeType, 2: MimeType, 3: MimeType, application/pdf: MimeType, application/x-google-chrome-pdf: MimeType, application/x-nacl: MimeType, application/x-pnacl: Mime_
    onLine: true
  > permissions: Permissions {}
  platform: "MacIntel"
  > plugins: PluginArray (0: Plugin, 1: Plugin, 2: Plugin, Chrome PDF Plugin: Plugin, Chrome PDF Viewer: Plugin, Native Client: Plugin, length: 3)
  > presentation: Presentation {defaultRequest: null, receivers: null}
  product: "Gecko"
  productSub: "20030107"
  > scheduling: Scheduling {}
  > serial: Serial {onconnect: null, ondisconnect: null}
  > serviceWorker: ServiceWorkerContainer {controller: null, ready: Promise, oncontrollerchange: null, onmessage: null, onmessageerror: null}
```

We can see that there are so many options available in the navigator, and one of them is **mediaDevices**, inside of which, we have **getUserMedia**!

Let's use it -

*Teacher guides the student in writing the code*

*Student writes the code*

```
let myStream;

navigator.mediaDevices
.getUserMedia({
  audio: true,
  video: true,
})
```

Here, we are using the **navigator.mediaDevices.getUserMedia()** function, and inside it, we are passing an object in which we state that we want to capture both **video** as well as **audio**!

Now, we have the user's audio and the video! We can use a **.then()** function to take the stream of audio and video that we are getting for the user, and save it into our variable called **myStream** now so we can use it later!

Let's do that -

*Teacher helps the student in writing the code*

*Student writes the code*

```
let myStream;

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

Awesome! Now, we have a stream that we can use as we see fit!

What do you think we need to do next?

Awesome! Now remember, just like we are displaying our stream in the video element, we will have to display other people's stream into the video element as well! What should we do in this case?

**ESR:**

Display this stream in the video element we created earlier

**ESR:**

Create a function!

Let's do that -

*Teacher helps the student in writing the code*

*Student writes the code*

```

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

function addVideoStream(video, stream) {
  video.srcObject = stream;
  video.addEventListener("loadedmetadata", () => {
    video.play();
    $("#video_grid").append(video)
  });
};

```

Here, we have created a function called **addVideoStream()** which takes 2 arguments -

1. **video** - The element in which we will display the stream
2. **stream** - The stream that will be displayed

Inside this function, just how we set the **myVideo.muted** to **true** earlier to make our video muted, we are setting it's **srcObject** to stream, so that it displays the stream!

Also, we are adding an event listener to the **video** element called **loadedmetadata**, which checks if the src object is loaded or not! This means that as soon as the stream is loaded, this event listener will fire it's function.

Inside the event listener, we are calling the **play()** function of the video element, and we are finally adding it to our **video\_grid** div element.

We are doing this to ensure that the video is only getting displayed, after the stream is completely loaded and ready.

Also, do note that we are calling this function inside the **then()** function of the **getUserMedia()** so that as soon as the stream is received, we can start displaying it.

Currently, it should only get displayed on our device but how do we test it?

That's right! For that, let's push our code to github first, and then deploy it through the dashboard! To push the code to github, open the cmd/terminal and in the project directory -

**git add -A**  
**git commit -m "video added"**  
**git push**

*Teacher helps the student in opening the command prompt/terminal, navigating to the project directory and running the commands and then deploying the code to Heroku from dashboard*

**ESR:**

By pushing this code on Heroku!

*Student follows the instructions*

#### Manual deploy

Deploy the current state of a branch to this app.

#### Deploy a GitHub branch

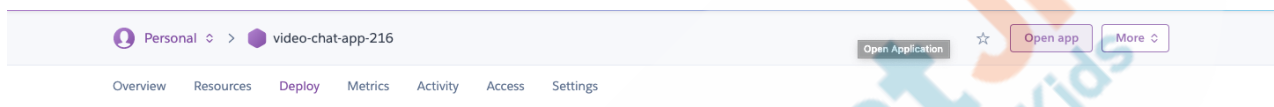
This will deploy the current state of the branch you specify below. [Learn more.](#)

Choose a branch to deploy

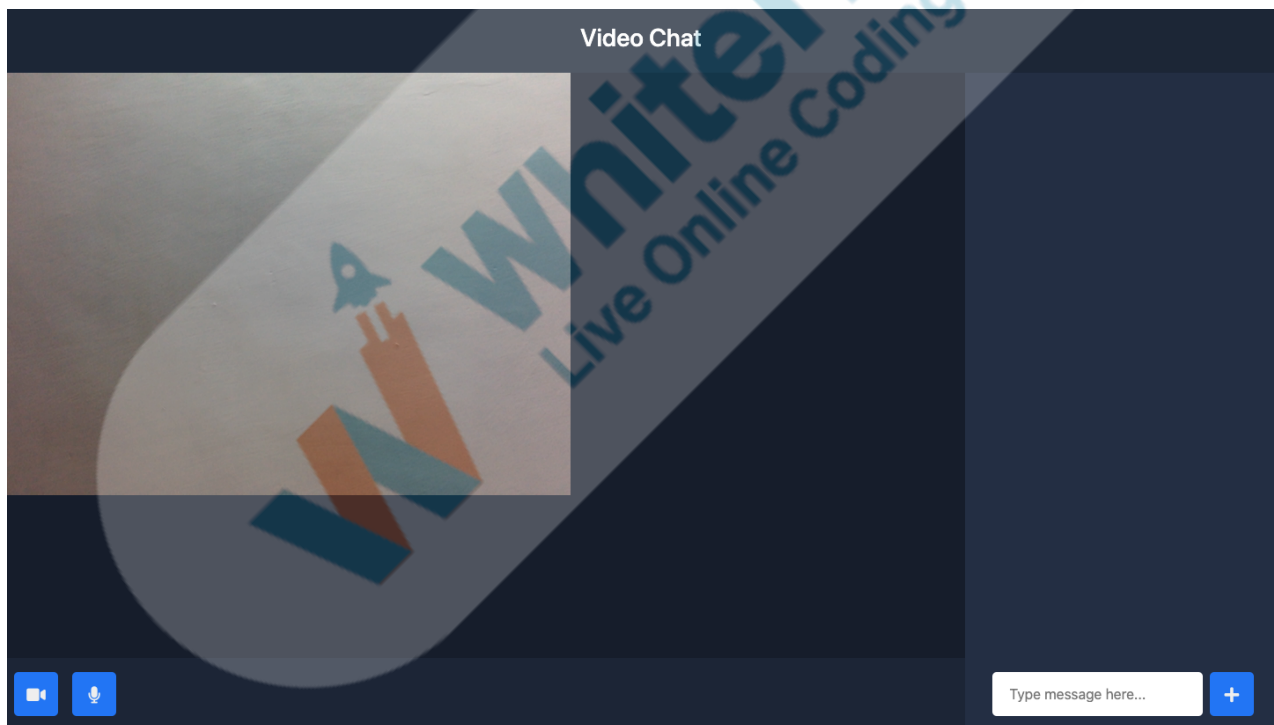
main

Deploy Branch

And there! Now, open your Heroku project on the browser and navigate to the deploy section and click on the **Open App** button -



And you should see the output!



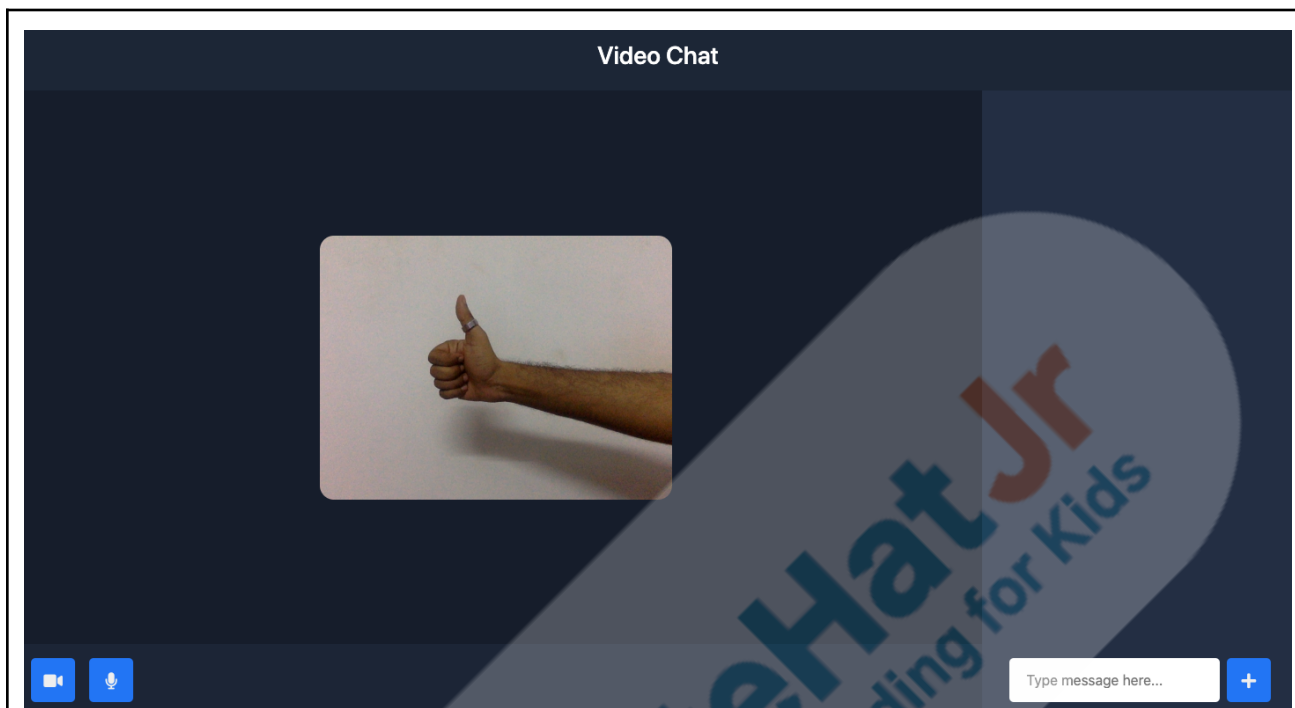
Let's add some styling to the video element to make it look better!

*Teacher helps the student in writing the code (can be copy pasted from the ref code)*

*Student writes the code*

```
#video_grid {  
  flex-grow: 1;  
  display: flex;  
  justify-content: center;  
  align-items: center;  
  padding: 1rem;  
  background-color: var(--main-dark);  
}  
  
video {  
  height: 300px;  
  border-radius: 1rem;  
  margin: 0.5rem;  
  width: 400px;  
  object-fit: cover;  
  transform: rotateY(180deg);  
  -webkit-transform: rotateY(180deg);  
  -moz-transform: rotateY(180deg);  
}
```

Let's push it again on Heroku and check the output -



Awesome! We are getting closer by the day! In the next class, we will complete our video chat functionality for this application!

**Teacher Guides Student to Stop Screen Share**

**WRAP UP SESSION - 5 Mins**




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

| Question  | Answer   |
|---|----------|
| <p>Webrtc stands for -----?</p> <p>A. Web real time chat<br/>B. Web server<br/>C. Web real time communication<br/>D. All of the above</p> | <b>A</b> |
| <p>What does rotateY do ?</p>   | <b>C</b> |

© 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>A. Rotate in both direction<br/>         B. Rotate in x direction<br/>         C. Rotate in y direction<br/>         D. None of the above</p>   |   |
|--|---|
| <p>What is the purpose of padding in the program?</p> <p>A. Create extra space<br/>         B. Add some space<br/>         C. Delete extra space<br/>         D. None of the above</p>   | <p><b>A</b></p>   |
| <p><b>End the quiz panel</b></p>   |   |
| <p style="text-align: center;"><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 streaming this audio and video to other peers using PeerJS</p>   | <p><i>Make sure you have given at least 2 Hats Off during the class for:</i></p> <div data-bbox="1031 1333 1323 1438"> <p>Creatively Solved Activities  +10</p> </div> <div data-bbox="1031 1459 1323 1564"> <p>Great Question  +10</p> </div> <div data-bbox="1031 1585 1323 1690"> <p>Strong Concentration  +10</p> </div> |



|   |  |
|---|--|
| <h2>Project Discussion</h2> <p>In Class 219, we learnt about what WebRTC is in depth, and how it is used to stream both video and audio. In this project, you'll work on completing some WebRTC code 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 been studying about WebRTC and how our own Audio and Video can be streamed and displayed by using Media devices, but she needs your help!</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-C218-ReferenceCode">https://github.com/pro-whitehatjr/PRO-C218-ReferenceCode</a> |
| Teacher Activity 2 | Reference Code      | <a href="https://github.com/pro-whitehatjr/PRO-C219-ReferenceCode">https://github.com/pro-whitehatjr/PRO-C219-ReferenceCode</a> |
| Student Activity 1 | Previous Class Code | <a href="https://github.com/pro-whitehatjr/PRO-C218-ReferenceCode">https://github.com/pro-whitehatjr/PRO-C218-ReferenceCode</a> |