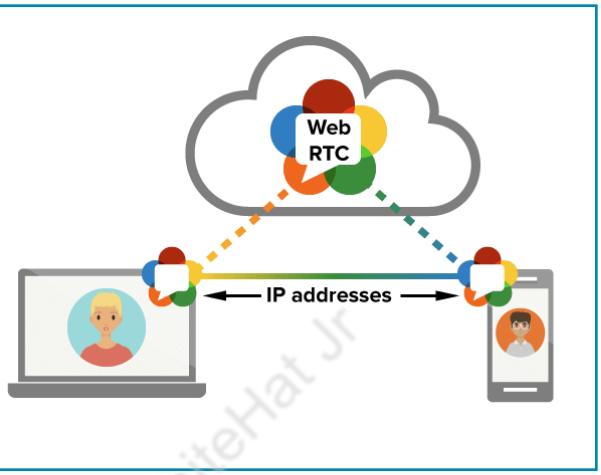
VIDEO CHAT APP - WebRTC



What is our GOAL for this MODULE?

The goal of this module is to learn about WebRTC and how Video and Audio can be gathered from the user on the browser and stream it to their fellow peers.

What did we ACHIEVE in the class TODAY?

- Learning about WebRTC
- Fetching Audio and Video of the users from the browser

Which CONCEPTS/CODING BLOCKS did we cover today?

- Understanding about WebRTC and it's functions.
- Fetching the audio and the video for the chat app from the user's browser.

The KEY CONCEPT

1. What is WebRTC?

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.

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.

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 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?

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!

How did we DO the activities?

1. In our output of the previous class, we have a box where the videos of the participants should be visible.

2. Let's check the HTML code of this.

```
<div class="row main">
    <div class="col-sm-12 col-md-12 col-lg-9 left-window">
        <div class="row">
            <div class="col-sm-12 col-md-12 col-lg-12" style="height: 81vh; background-color: #171e2a;">
                <!-- Video Streams -->
            </div>
```

3. Here, we can notice that there is a div container in which we will display the video streams. Let's give this ***div*** and ***id*** called **video_grid** -

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

This is to identify the div we want to add the streams to in our script.

4. Code for the browser to take video and audio. Open the ***script.js*** file -
Here, after asking the user to enter their name as a ***prompt***, we create a new ***video*** element.

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

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

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

6. To fetch our video and audio stream there is a special function in JavaScript, known as ***getUserMedia()***.

```
let myStream;

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

7. 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 myStream;

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

8. Create 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 the function, set srcObject to stream to display the stream.

Also, add an event listener to the video element called loadedmetadata which checks if the src object is loaded or not. Inside the event listener, we will call the ***play()*** function of the video element, and we will add it to the ***video_grid*** div element.

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

Add some styling to the code.

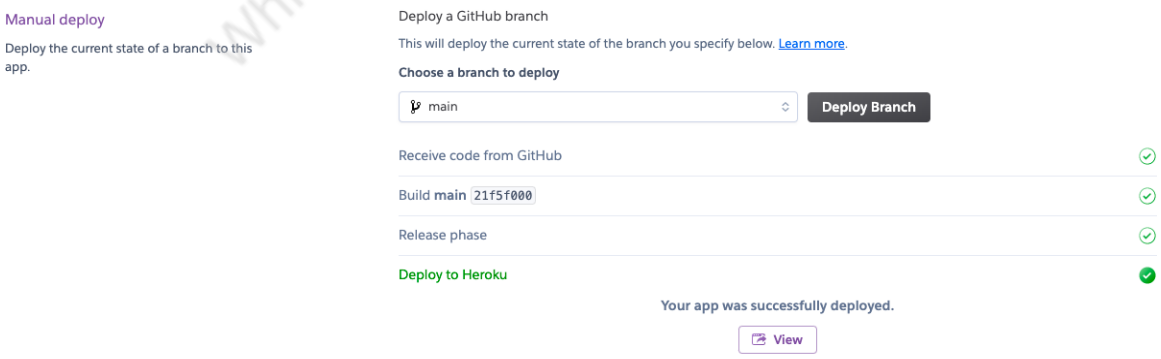
```css
#video_grid {
  flex-grow: 1;
  display: flex;
  justify-content: center;
  align-items: center;
  padding: 1rem;
  background-color: var(--main-dark);
}

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);
}
```

9. To test it, push the code on Heroku.

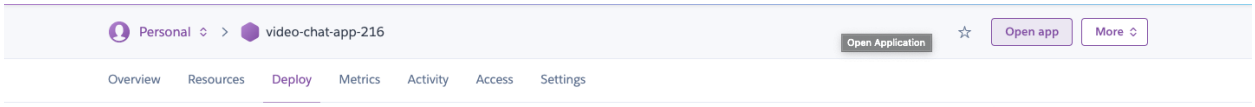
   Open the command prompt/terminal and navigate to the directory in which we have the project, and then run the following commands -

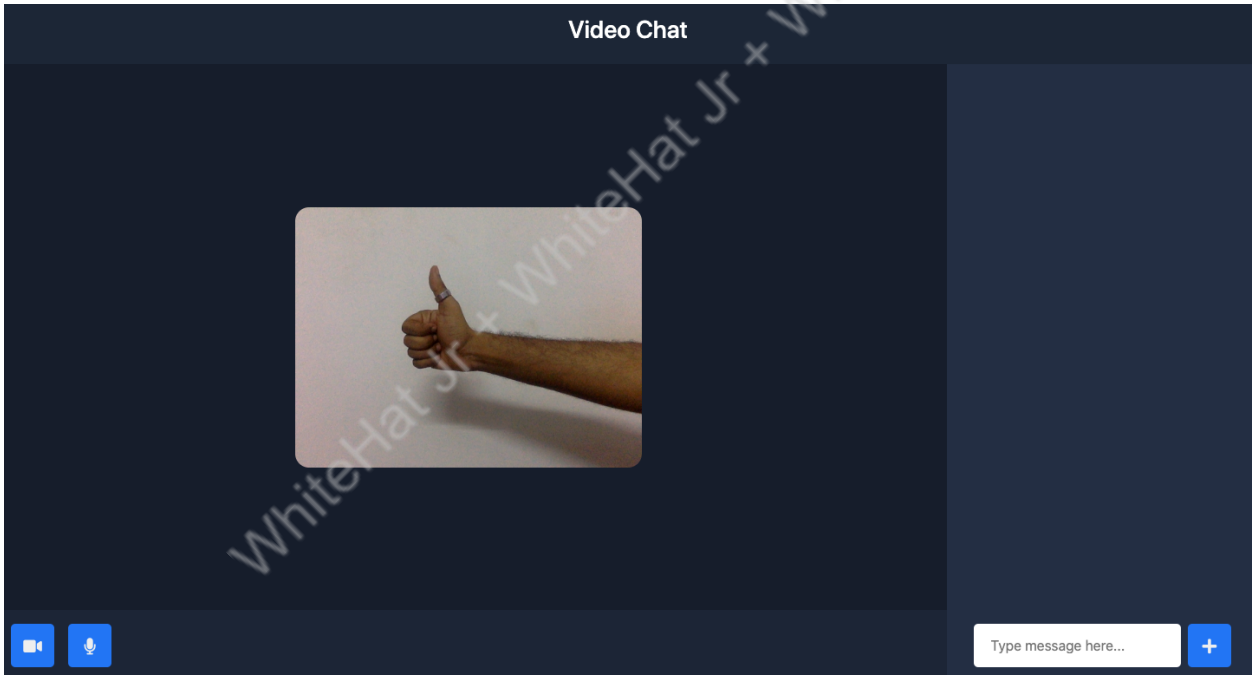
   git add -A
   git commit -m "video added"
   git push

   And finally from Heroku Dashboard, deploy the code -



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





What's NEXT?

In the next class we will _____

EXTEND YOUR KNOWLEDGE

You can learn more about WebRTC here.