**Pre-Project Setup**

Here’s what you’ll need:

* **NodeJS:**Visit the official [Node.js website](https://nodejs.org/en/) to download and install Node.
* **NPM:**NPM program gets installed on your computer when you install Node.js.

**Project Setup**

*All the code for this project can be found in the GitHub Repo*

1. Create an empty directory named video-chat-app.
2. Open up your console, navigate to our new directory, and run npm init.
3. Fill out the required information to initialize our project.
4. Within video-chat-app the directory, and run npm install express ejs socket.io uuid peer. This will install all the dependency that we need to build this application.
5. And also as a dev dependency, we will install [Nodemon](https://nodemon.io/). runnpm install — dev nodemon. This will install nodemon as a dev dependency.
6. Create a file named server.js — this file will keep all our server-side logic

Now that we have our project setup, we can start building!

**Creating our Server (with Express JS)**

The first thing we need to do is get our server up and running. We’re going to use Express to accomplish this. Express is a minimalist web framework for Node.js — Express makes it very easy to create and run a web server with Node.

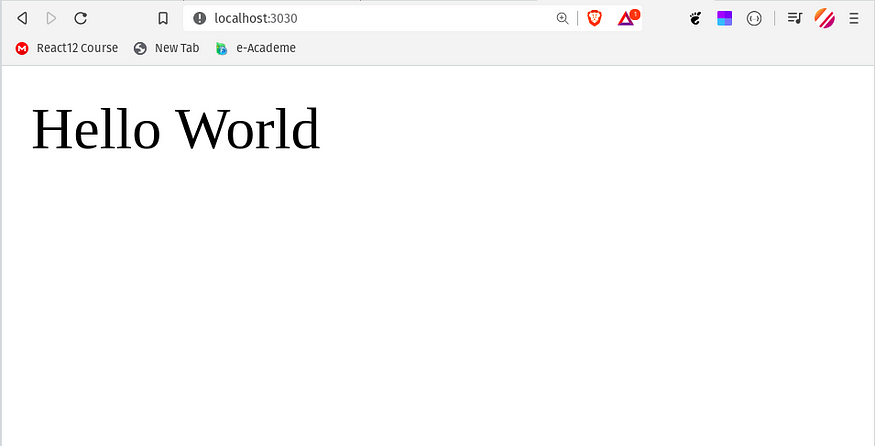
Let’s create a boilerplate Express starter app file.

**// server.js**  
const express = require(“express”);  
const app = express();  
const server = require(“http”).Server(app);  
app.get(“/”, (req, res) => {  
 res.status(200).send(“Hello World”);  
});  
server.listen(3030);

Now we have our server running, We can test our server by running:

> nodemon server.js

Now open your browser and visit: localhost:3000 and you should see Hello World.



**Creating our first View**

Instead of responding with text when someone visits our root route, we’d like to respond with an HTML file. For this, we’ll be using EJS (Embedded JavaScript). EJS is a templating language.

In order to use EJS in Express, we need to set up our template engine. To setup add this line of code in server.js file.

app.set(‘view engine’, ‘ejs’)

EJS is accessed by default in the views directory. Now create a new folder named views in your directory. Within that views folder, add a file named room.ejs. Think of our room.ejs file as an HTML file for now.

Here’s what our file structure looks like so far:

|-- video-chat-app  
 |-- views  
 |-- room.ejs  
 |-- package.json  
 |-- server.js

Now we have our boilerplate forroom.ejsthe file. Now we will add our HTML code to room.ejs thefile.

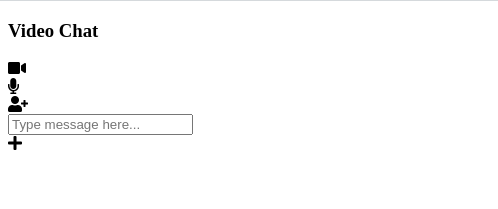
Once you copy the above code, we need to replace our app. js get code:

app.get(‘/’, function (req, res) {  
 **// OLD CODE**  
 res.status(200).send("Hello World");  
})

Above is the old code where we send the text ‘Hello World!’ to the client. Instead, we want to send our room.ejs file:

app.get(‘/’, function (req, res) {  
 **// NEW CODE**  
 res.render(‘room’);  
})

Now open your browser and visit: localhost:3030 and you should see our room.ejs file being displayed!



**Adding a CSS File**

This doesn't look good right? That is because we don't have any stylesheet files in our project. So let's add some CSS.

We’ll need to add a new folder to our project called public. Within that folder create a file named style.css and script.js. Here’s our new file structure:

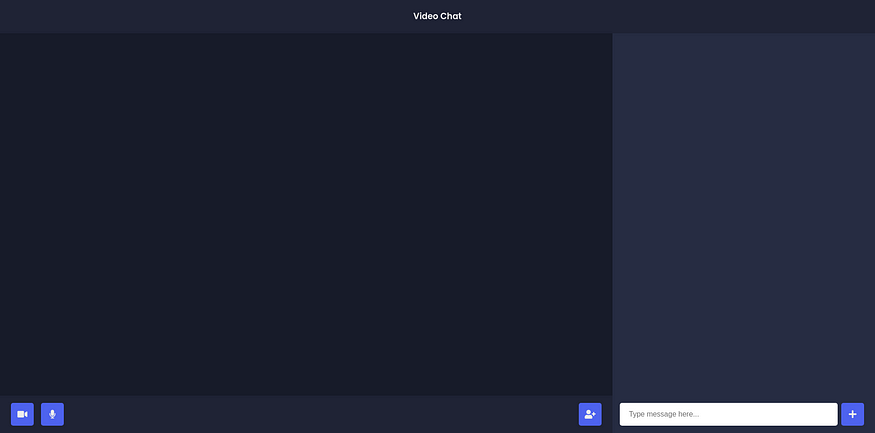
|-- weather-app  
 |-- views  
 |-- index.ejs  
 |-- public  
 |-- style.css  
 |-- script.js  
 |-- package.json  
 |-- server.js

Express won't allow access to this file by default, so we need to expose it with the following line of code:

app.use(express.static(‘public’));

This code allows us to access all of the static files within the ‘public’ folder.  
Finally, we need our CSS. Since this isn’t a CSS course, I’m not going to be going over the specifics, but if you’d like to use my CSS, you can copy it from [**here**](https://github.com/itstaranarora/video-chat-v1/blob/master/public/style.css)**.**

Once you added CSS. you can visit: localhost:3030/ you will notice your application looks a little nicer.



**Setting up our Rooms**

By now, your server.jsfile should look like this:

We have one get route, and then we create our server. However, for our application to work, Whenever a new user visits our default route we will redirect him to a unique URL. We will use [**uuid**](https://www.npmjs.com/package/uuid) library to create a random unique URL for each room.

UUID is a javascript library that allows us to create unique Ids. In our application, we will use uuid version 4 to create our unique URL. But first let's import uuid in our server.js file.

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

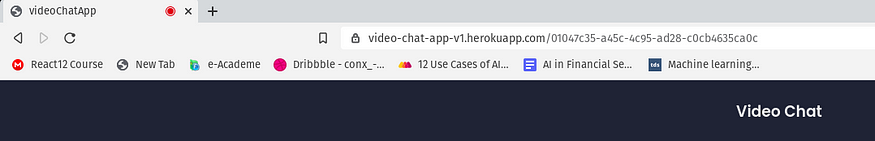
Now, We will use [uui*d*](https://www.npmjs.com/package/uuid) library to create a random unique id for each room. and we will redirect our user to that room.

app.get(“/”, (req, res) => { res.redirect(`/${uuidv4()}`);});

And Before we test this I also wanted to add a view for every unique room and we will pass the current URL to that view.

app.get(“/:room”, (req, res) => { res.render(“room”, { roomId: req.param.room });});

We have passed**roomId** to room.ejs and with this we are done setting our Rooms. and now if you visit localhost:3030. you will be redirected to a unique URL.



**Adding Current User Video**

We will work on our script.js file that we have created earlier. script.js will contain all our client-side code.

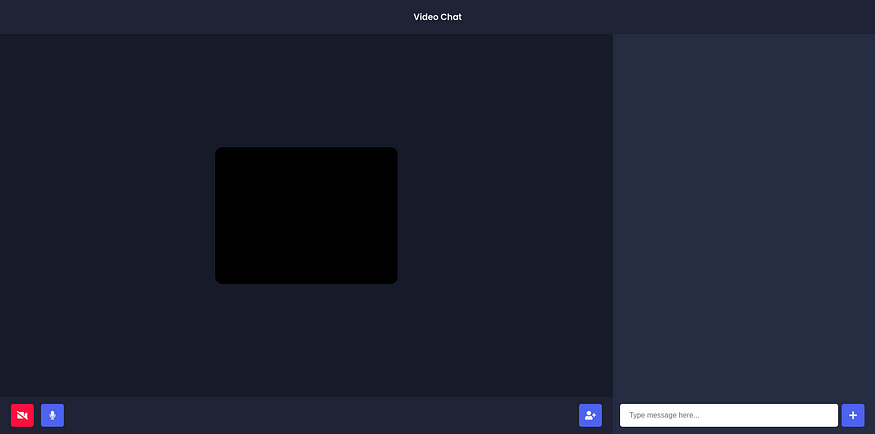
So here what we need to do, we need to get the video stream aka user media, and then we will add that stream to a video element.

let myVideoStream;  
const videoGrid = document.getElementById("video-grid");  
const myVideo = document.createElement("video");  
myVideo.muted = true;  
navigator.mediaDevices.getUserMedia({  
 audio: true,  
 video: true,  
})  
.then((stream) => {  
 myVideoStream = stream;  
 addVideoStream(myVideo, stream);  
});

Now we will create addVideoStream function. which will add the stream to the video element.

const addVideoStream = (video, stream) => {  
 video.srcObject = stream;  
 video.addEventListener("loadedmetadata", () => {  
 video.play();  
 videoGrid.append(video);  
 });  
};

This code will add a user stream to the video element. you can test this by visiting localhost:3030 and you will see your video popup in the video element



**5. Adding the ability to allow others to stream their video**

Now it’s time to use Socket.io and PeerJS. For those who don’t know Socket.io allows us to do real-time communication. and PeerJS allow us to implement WebRTC.

First, we will import socket.io and peerjs in our server.js file and listen for join-room event.







# ****6. Building User Interface****

We are done with our video part. Now let’s do some styling. but first, let’s add some content to **room.ejs** file. (Add font awesome CDN inside your head tag)





