



**BRAINWARE UNIVERSITY**

**A Project Report  
on  
Group I.O.**

***Submitted by:***

***SIDDHANTH DAS***

***DEBANKOJIT ROY***

***TABNA SHAHID***

***SAMIR GHOSH***

## **ACKNOWLEDGEMENT**

The successful completion of any project is not possible without a well-informed support. Likewise, in the completion of this project without the constant guidance and encouragement of our Industrial Training instructor, Mr. Surajit Das it wouldn't have been possible. His patient and constructive suggestions helped us to put this project together for which we are highly grateful.

# TABLE OF CONTENTS

<b>1.</b>	<b>Introduction</b>	<b>4</b>
	<b>1.1. Objective</b>	
	<b>1.2. Significance of Node.js</b>	
<b>2.</b>	<b>Overview</b>	<b>5-12</b>
	<b>2.1. Technologies used</b>	
	<b>2.2. How to make a video call app in node.js?</b>	
<b>3.</b>	<b>Dependencies in Detail</b>	<b>13-23</b>
<b>4.</b>	<b>Conclusion</b>	<b>24</b>
<b>5.</b>	<b>Future Scope</b>	<b>24</b>
<b>6.</b>	<b>References</b>	<b>25</b>

## **1. Introduction**

Teleconferencing or Video Chatting, is a method of using technology to bring people and ideas “together” despite of the geographical barriers. The technology has been available for years but the acceptance is gradually increasing.

Our project is an example of a video chat server. It is made using NodeJS which is a server-side JavaScript run-time environment. To start chatting client should get connected to server where they can create rooms and invite people to video chat with them.

### **1.1. Objective:**

To create a simple application that allows us to stream audio and video to the connected device – a basic video chat app. It will be a multi-user Web Application, which the client will be able to join via a randomly generated URL.

### **1.2. Significance of Node.js:**

To make a remote connection between two or more devices we need a server. In this case, we need a server that handles real-time communication. You know that Node.js is built for real-time scalable applications. To develop two-way connection apps with free data exchange, you would probably use Web-Sockets that allows opening a communication session between a client and a server. Requests from the client are processed as a loop, more precisely – the event loop, which makes Node.js a good option because it takes a “non-blocking” approach to serve requests and thus, achieves low latency and high throughput along the way.

## 2. Overview

### 2.1. Technologies used

- **Node.js** - Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript, and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.
- **HTML** - HTML stands for Hyper Text Markup Language. HTML describes the structure of a Web page. HTML consists of a series of elements. HTML elements tell the browser how to display the content. HTML elements are represented by tags.
- **CSS** - Cascading Style Sheets, fondly referred to as CSS, is a simple design language intended to simplify the process of making web pages presentable. CSS handles the look and feel part of a web page.
- **Bootstrap** - Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains CSS- and JavaScript-based design templates for typography, forms, buttons, navigation, and other interface components.

## 2.2. How to make a video call app in node.js?

For making a video call app, it is required that each and every client send their video and audio stream to all the other clients. So, for this purpose, we are using Peer.js and for the communication between the clients and the server we are using WebSocket i.e. Socket.io.

### ➤ Prerequisites:

1. **Node.js**: It is an open-source JavaScript Back-End technology. It has a package manager called **npm**– **Node package manager** which installs different packages very easily.
2. **Express.js**: It is a node.js server framework.
3. **Socket.io**: It helps us to create a real-time bi-direction event-based communication between the server and the client.
4. **Peer.js**: It helps us to send and receive the audio and video streams of the other clients.

- **Setting up the Environment**: This is the very first step, here we are creating and initializing a new repository.

```
$ mkdir VideoCallApp
```

```
$ cd VideoCallApp
```

```
$ npm init
```

Now, the next step is to install the required packages for our VideoCallApp.

- **Express**: It is the server-based framework for node.js
- **ejs**: It is a simple templating language that lets you generate HTML markup with plain JavaScript.

- **Socket.io:** It manages the Websocket for event-based communication.
- **Nodemon (optional):** It automatically restarts the server when you save your project files.
- **uuid module:** This module is used to generate a unique id. This will be used in this project.

➤ **Installing the required modules:**

*\$ npm install express*

*\$ npm install ejs*

*\$ npm install socket.io*

*\$ npm install nodemon*

Now, we all set for the implementation part.

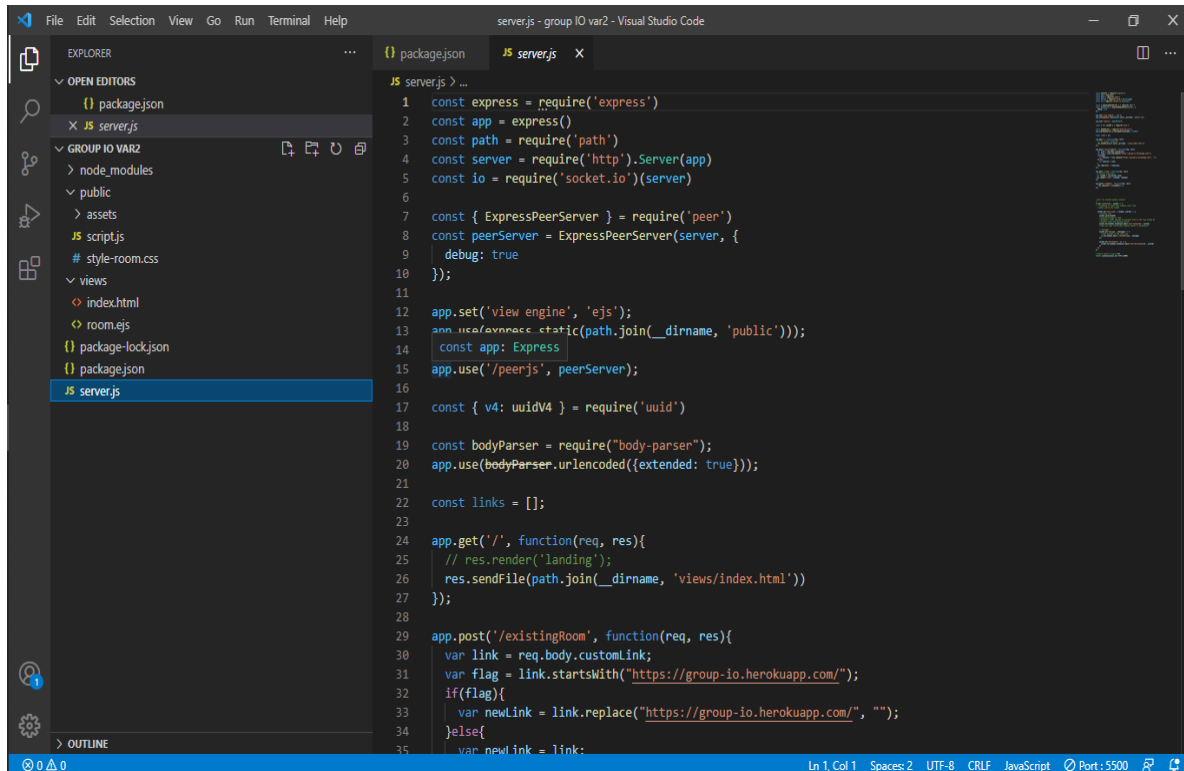
The code for serving the UI HTML file to the client will be written in the **server.js** file.

Our service will start with a home page, named as index.html, then upon providing the URL, a new window will open (the page served would be called **room.ejs**) where all the connected users will be displayed, in video-meeting type interface.

➤ **Create the `server.js` file:**

We import the required modules.

Set the directory path name to public from where all the assets of our HTML and ejs file will be served from.



The screenshot shows the Visual Studio Code interface with a project named "serverjs - group IO var2". The Explorer sidebar on the left shows the file structure: "package.json", "server.js", "node\_modules", "public" (containing "assets", "scripts", "style-room.css", "views" (containing "index.html", "room.ejs")), "package-lock.json", and "package.json". The "server.js" file is selected and open in the editor. The code in "server.js" is as follows:

```
1 const express = require('express')
2 const app = express()
3 const path = require('path')
4 const server = require('http').Server(app)
5 const io = require('socket.io')(server)
6
7 const { ExpressPeerServer } = require('peer')
8 const peerServer = ExpressPeerServer(server, {
9   debug: true
10 });
11
12 app.set('view engine', 'ejs');
13 app.use(express.static(path.join(__dirname, 'public')));
14 const app: Express
15 app.use('/peerjs', peerServer);
16
17 const { v4: uuidv4 } = require('uuid')
18
19 const bodyParser = require("body-parser");
20 app.use(bodyParser.urlencoded({extended: true}));
21
22 const links = [];
23
24 app.get('/', function(req, res){
25   // res.render('landing');
26   res.sendFile(path.join(__dirname, 'views/index.html'))
27 });
28
29 app.post('/existingRoom', function(req, res){
30   var link = req.body.customlink;
31   var flag = link.startsWith("https://group-io.herokuapp.com/");
32   if(flag){
33     var newLink = link.replace("https://group-io.herokuapp.com/", "");
34   }else{
35     var newLink = link;
```

Then we write the code for the landing pages on our server and the subsequent pages that will follow



```

17 const { v4: uuidV4 } = require('uuid')
18
19 const bodyParser = require("body-parser");
20 app.use(bodyParser.urlencoded({extended: true}));
21
22 const links = [];
23
24 app.get('/', function(req, res){
25   // res.render('landing');
26   res.sendFile(path.join(__dirname, 'views/index.html'))
27 });
28
29 app.post('/existingRoom', function(req, res){
30   var link = req.body.customLi Follow link (ctrl + click)
31   var flag = link.startsWith("https://group-io.herokuapp.com/");
32   if(flag){
33     var newLink = link.replace("https://group-io.herokuapp.com/", "");
34   }else{
35     var newLink = link;
36   }
37   res.redirect('//' + newLink);
38 });
39
40 app.get('/:room', function(req, res){
41   // console.log(links)
42   var roomId = req.params.room;
43   res.render('room', {roomId: roomId})
44 });
45
46 app.post('/newRoom', function(req, res){
47   res.redirect('/?' + uuidV4());
48 });
49
50

```

Now, for socket.io we need to write some more code. Here, we added code for socket.io and we just change *app.listen()* to *server.listen()* methods.

```

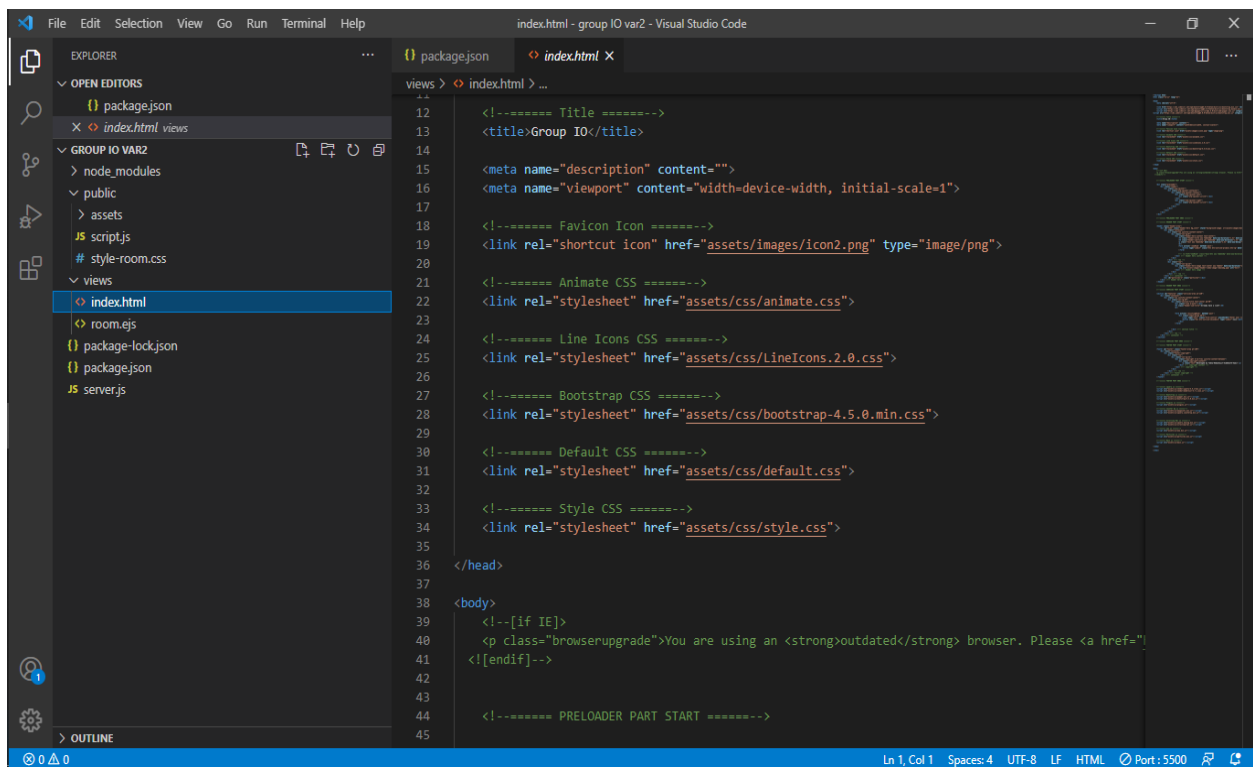
51
52
53
54 //will run anytime anyone connects
55 //
56 io.on('connection', socket => {
57   //listening to event when someone joins room
58   //sends userid and roomId
59
60   socket.on('join-room', (roomId, userId) => {
61     //joining room
62     socket.join(roomId)
63     //sending message to room
64     //broadcast sends message to everyone else in the room except me
65     //another event and passing userid
66     socket.to(roomId).broadcast.emit('user-connected', userId)
67     //this will get called when someone leaves or disconnects
68
69     // messages
70     socket.on('message', (message) => {
71       //send message to the same room
72       io.to(roomId).emit('createMessage', message)
73     });
74
75     socket.on('disconnect', () => {
76       socket.to(roomId).broadcast.emit('user-disconnected', userId)
77     })
78   })
79 })
80
81 //starts server on port 3000
82 server.listen(process.env.PORT || 3000)

```

At this point, we are all set for the client-side development.

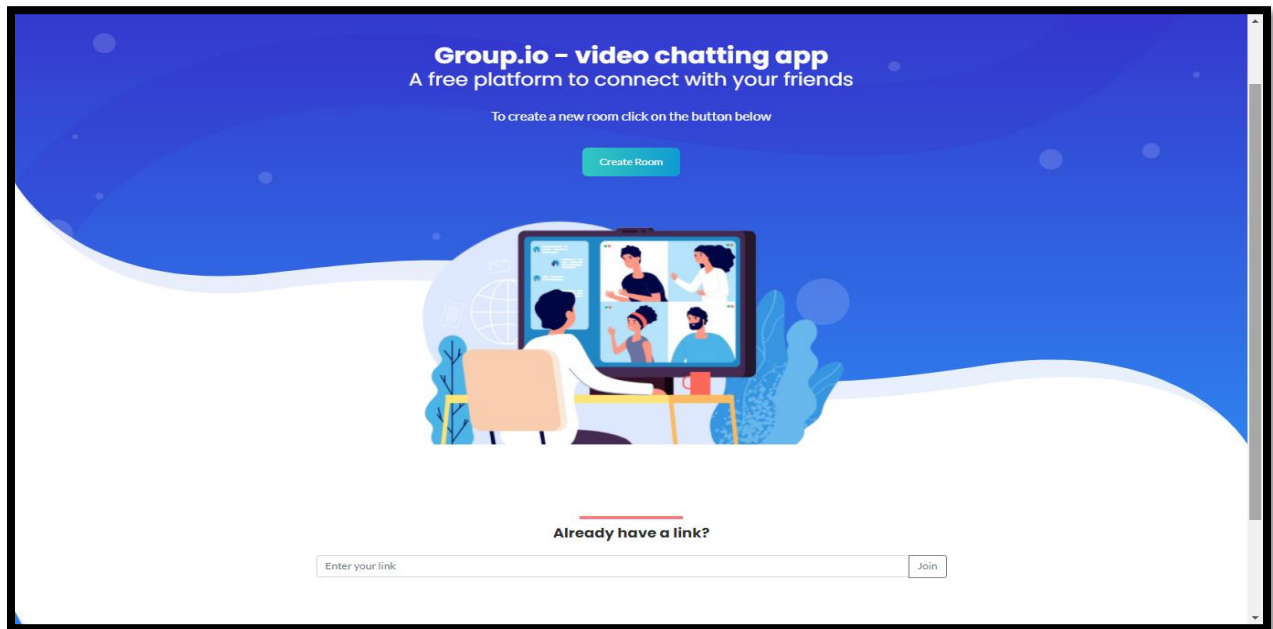
Now in **views** create an **index.html** file which contains all html code.

Here, we added two script files, first one is our main **index.html** file and the second one is a **room.ejs** file which will act as the interface for the video meeting.



```
12 <!--===== Title =====>
13 <title>Group IO</title>
14
15 <meta name="description" content="">
16 <meta name="viewport" content="width=device-width, initial-scale=1">
17
18 <!--===== Favicon Icon =====>
19 <link rel="shortcut icon" href="assets/images/icon2.png" type="image/png">
20
21 <!--===== Animate CSS =====>
22 <link rel="stylesheet" href="assets/css/animate.css">
23
24 <!--===== Line Icons CSS =====>
25 <link rel="stylesheet" href="assets/css/LineIcons.2.0.css">
26
27 <!--===== Bootstrap CSS =====>
28 <link rel="stylesheet" href="assets/css/bootstrap-4.5.0.min.css">
29
30 <!--===== Default CSS =====>
31 <link rel="stylesheet" href="assets/css/default.css">
32
33 <!--===== Style CSS =====>
34 <link rel="stylesheet" href="assets/css/style.css">
35
36 </head>
37
38 <body>
39 <!--[if IE]>
40 <p class="browserupgrade">You are using an <strong>outdated</strong> browser. Please <a href="
41 <![endif]>-->
42
43
44 <!--===== PRELOADER PART START =====>
45
```

The homepage will look like this:

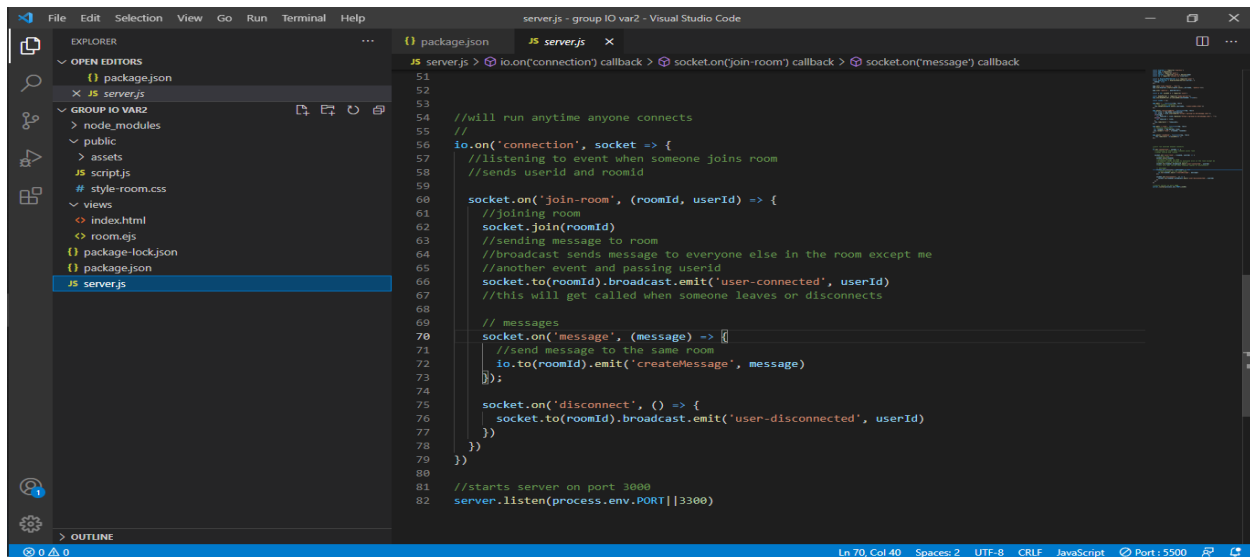


- The '*Create room*' option will redirect the user to a new room, with only the user, and they can use the URL for other people to join.
- The input field below, is where a user can insert the URL and join an existing room.

### ➤ Connecting the users:

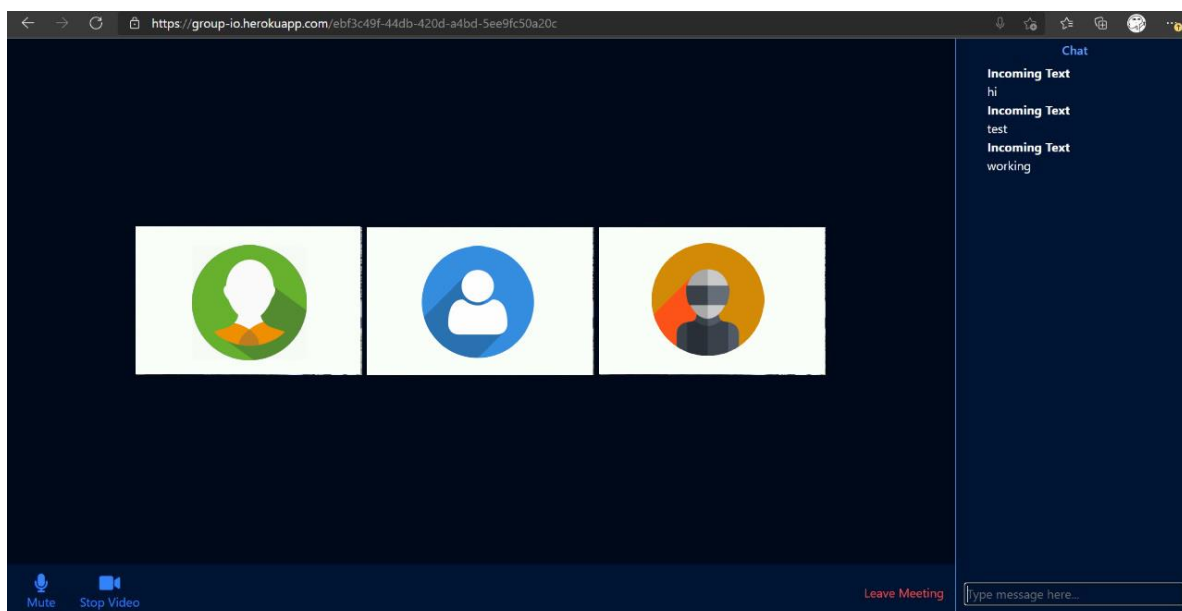
So, whenever the new user will get connected, they will get a unique id through the peer.js and after that it emits a socket event for the server i.e newUser.

So to handle this event we need to add some more code in our server-side(in the server.js file):



```
51
52
53
54 //will run anytime anyone connects
55 //
56 io.on('connection', socket => {
57   //listening to event when someone joins room
58   //sends userid and roomid
59
60   socket.on('join-room', (roomId, userId) => {
61     //joining room
62     socket.join(roomId)
63     //sending message to room
64     //broadcast sends message to everyone else in the room except me
65     //another event and passing userid
66     socket.to(roomId).broadcast.emit('user-connected', userId)
67     //this will get called when someone leaves or disconnects
68
69     // messages
70     socket.on('message', (message) => {
71       //send message to the same room
72       io.to(roomId).emit('createMessage', message)
73     });
74
75     socket.on('disconnect', () => {
76       socket.to(roomId).broadcast.emit('user-disconnected', userId)
77     })
78   })
79
80 //starts server on port 3000
81 server.listen(process.env.PORT||3300)
```

After the users are connected, they will appear in the following interface in the connected room:



### 3. Dependencies in Detail

#### ❖ Sockets.IO:

Earlier, websites used to reload every-time a resource was requested. This introduced unnecessary delays which increased average wait time. Often users had to wait for minutes to fetch a particular page or file. Real-time applications (Instant messenger, Online gaming, push notification etc), on the other hand, are those applications which run within a given time-slot such that user is presented with immediate and up-to-date copy of the resource. Latency in these applications is kept as low as possible to give smooth and consistent user experience. Socket.IO is one such JavaScript library that programmers use in developing real-time “Web Applications”.

#### Why Sockets.IO :

Most of the applications on Internet today are based on Client-Server architecture. A client is someone who requests something from a Server. A Server, based on the request, responds with appropriate results. These two entities are completely different from each other because of the nature of tasks they perform. A browser is a perfect example of client application. Clients on browsers usually communicate to Servers via HTTP requests and responses. The problem with this communication is that either a request or a response can be sent at a time. For understanding, think of it as a half-duplex link. Also, HTTP headers contain lots and lots of redundant information which is useless once a connection between client and server is made. Sockets on the other hand work on transport layer of Network Stack. There are not many redundant fields thus increase the efficiency of information transfer over web.

Socket.IO works on the same concept and enables bi-directional communication between web clients and servers. For handling them separately and efficiently, it consists of two parts;

- a JavaScript client library that runs on browsers.
- a Node.js server

Socket.IO relies on Engine.IO, which is the implementation of the transport-based cross-browser/cross-device bi-directional communication layer. It brings in the following features to Socket.IO;

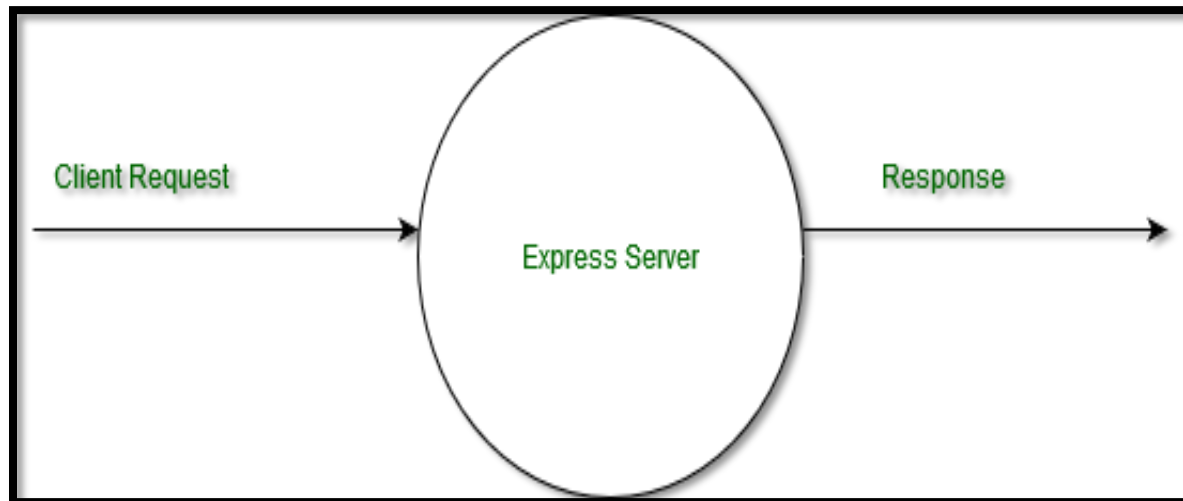
- **Reliability:** It can establish connection even in the presence of proxies, load-balancers, personal firewalls, and antivirus softwares.
- **Auto-reconnect Support:** Unless mentioned explicitly in the code, the client library will try to reconnect forever, until the server is available again.
- **Disconnection detection:** It allows both the server and the client to know when the other one is not responding anymore.
- **Multiplexing support:** It allows to have several communication channels on a same underlying connection.
- **Binary Streaming support:** It also allows emitting any serializable binary data like Array Buffer, Blobs, etc.

#### ❖ Express.js:

Express.js is a routing and Middleware framework for handling the different routing of the webpage and it works between the request and response cycle.

#### Framework:

It is known to be a skeleton where the application defines the content of the operation by filling out the skeleton. For Web development, there is python with Django, java with spring, and For Web development in we have Node.js with Express.js in node.js there is an HTTP module by which we can create an only limited operatable website or web application. In general, the real working of any web application or website is that it is capable to handle any kind of request. Requests may be post, get, delete, and many more like a request for an image, video, etc that's why Express.js is used as a Framework for Node.js.



There are lots of middleware functions in Express.js like [Express.js app.use\(\) Function](#) etc.

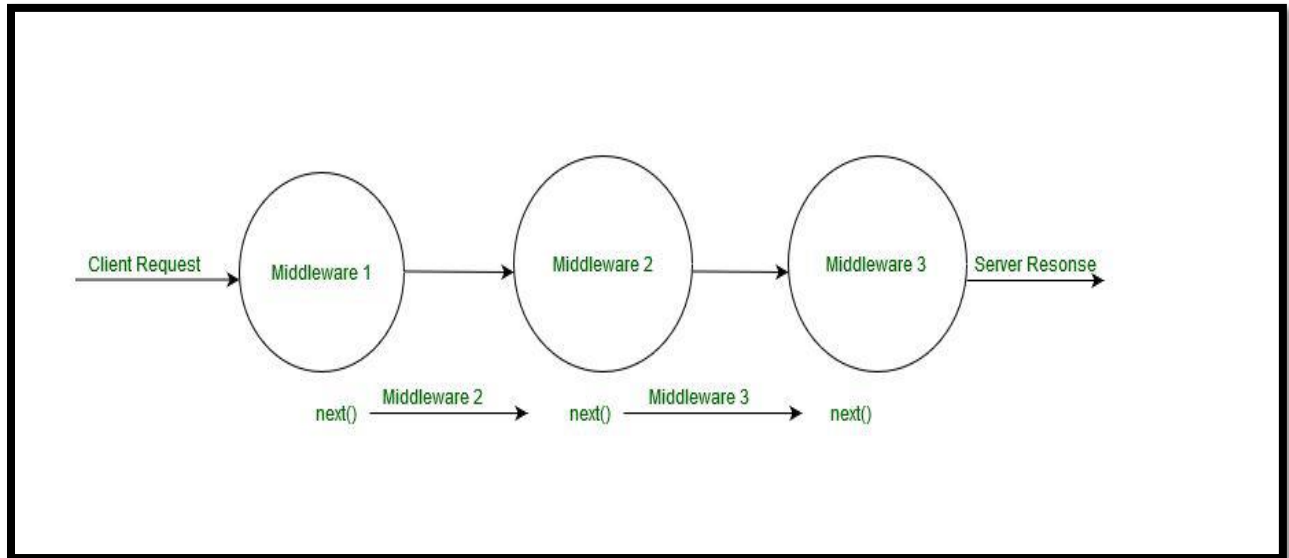
### Syntax:

```
app.use(path,(req,res,next))
```

**Parameters:** It accepts the two parameters as mentioned above and described below:

- **path:** It is the path for which the middleware function is being called. It can be a string representing a path or path pattern or a regular expression pattern to match the paths.
- **callback:** It is the callback function that contains the request object, response object, and next() function to call the next middleware function if the response of current middleware is not terminated. In the second parameter, we can also pass the function name of the middleware.

### The working cycle of multiple Middleware:



### Benefits of using Express.js Middleware:

1. We generally use `http.createServer()` to create a server and performs request and response according to the information, but we cannot check what type of request made by the client so that we can perform operations according to the request.
2. Express.js contains multiple methods to handle all types of requests rather than work on a single type of request as shown below:
  - `Express.js req.get()` Method: This method is used when get request is done by the client for eg: Redirecting another webpage requests etc
  - `Express.js req.post()` Method: This method is used when post requests are done by the client for eg uploading documents etc.
  - `Express.js req.delete()` Method: This method is used when a delete request is done by the client it is mainly done by the admin end for e.g. deleting the records from the server.
  - `Express.js req.put()` Method: This method is used when update requests are done by the client to update the information over the website.



3. Easy to connect with databases such as MongoDB, MySQL.
4. Easy to serve static files and resources we can easily serve HTML documents using express.js.
5. There are several other benefits of using Express.js like handling multiple get requests on a single webpage that means Allows you to define multiple routes of your application based on HTTP methods and URLs.

#### ❖ EJS:

EJS or Embedded JavaScript Templating is a templating engine used by Node.js. Template engine helps to create an HTML template with minimal code. Also, it can inject data into HTML template at the client side and produce the final HTML. EJS is a simple templating language which is used to generate HTML markup with plain JavaScript. It also helps to embed JavaScript to HTML pages.

To begin with, using EJS as templating engine we need to install EJS using given command:

```
npm install ejs --save
```

#### ❖ Peer.JS:

PeerJS wraps the browser's WebRTC implementation to provide a complete, configurable, and easy-to-use peer-to-peer connection API. Equipped with nothing but an ID, a peer can create a P2P data or media stream connection to a remote peer.

##### **Setup:**

##### Include the library

```
<script src="https://unpkg.com/peerjs@1.3.1/dist/peerjs.min.js"></script>
```

##### Create a peer

```
var peer = new Peer();
```

PeerJS uses PeerServer for session metadata and candidate signaling. You can also **run your own PeerServer** if you don't like the cloud.

We're now ready to start making connections!

## Usage:

Every Peer object is assigned a random, unique ID when it's created.

```
peer.on('open', function(id) {  
  console.log('My peer ID is: ' + id);  
});
```

When we want to connect to another peer, we'll need to know their peer id. You're in charge of communicating the peer IDs between users of your site. Optionally, you can pass in your own IDs to the **Peer constructor**.

## Data connections:

Start a data connection by calling `peer.connect` with the peer ID of the destination peer. Anytime another peer attempts to connect to your peer ID, you'll receive a connection event.

### Start connection

```
var conn = peer.connect('dest-peer-id');
```

### Receive connection

```
peer.on('connection', function(conn) { ... });
```

`peer.connect` and the callback of the `connection` event will both provide a `DataConnection` object. This object will allow you to send and receive data:

```
conn.on('open', function() {  
  // Receive messages  
  conn.on('data', function(data) {  
    console.log('Received', data);  
  });  
  
  // Send messages  
  conn.send('Hello!');  
});
```

## Video/audio calls:

Call another peer by calling *peer.call* with the peer ID of the destination peer. When a peer calls you, the *call* event is emitted.

Unlike data connections, when receiving a *call* event, the call must be answered or no connection is established.

Start call	Answer call
<pre>// Call a peer, providing our mediaStream var call = peer.call('dest-peer-id',   mediaStream);</pre>	<pre>peer.on('call', function(call) {   // Answer the call, providing our mediaStream   call.answer(mediaStream); });</pre>

## ❖ Nodemon:

The nodemon Module is a module that develop node.js based applications by automatically restarting the node application when file changes in the directory are detected. Nodemon does not require any change in the original code and method of development.

### Advantages of Using nodemon Module:

1. It is easy to use and easy to get started.
2. It does not affect the original code and no instance require to call it.
3. It helps to reduce the time of typing the default syntax node <file name> for execution again and again.

**Installation:** Install the module using the following command:

```
npm install -g nodemon
```

After installing the module, you can check the current version of the module by typing on console as shown below:

```
nodemon version
```

```
PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL
Anil@skynet MINGW64 ~/Desktop/Server Side Course/node-express (master)
$ nodemon version
[nodemon] 2.0.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node version index.js`
prompt: username: (node:7632) Warning: Accessing non-existent property 'padLevels' of module exports inside circular dependency
(Use `node --trace-warnings ...` to show where the warning was created)
```

## Usage:

1. The nodemon wraps your application, so you can pass all the arguments you would normally pass to your app:

*`nodemon [your node app]`*

2. Options available for nodemon are shown below:

*`nodemon -h`*

```
Anil@skynet MINGW64 ~/Desktop/Server Side Course/checking
$ nodemon -h
Usage: nodemon [options] [script.js] [args]

Options:

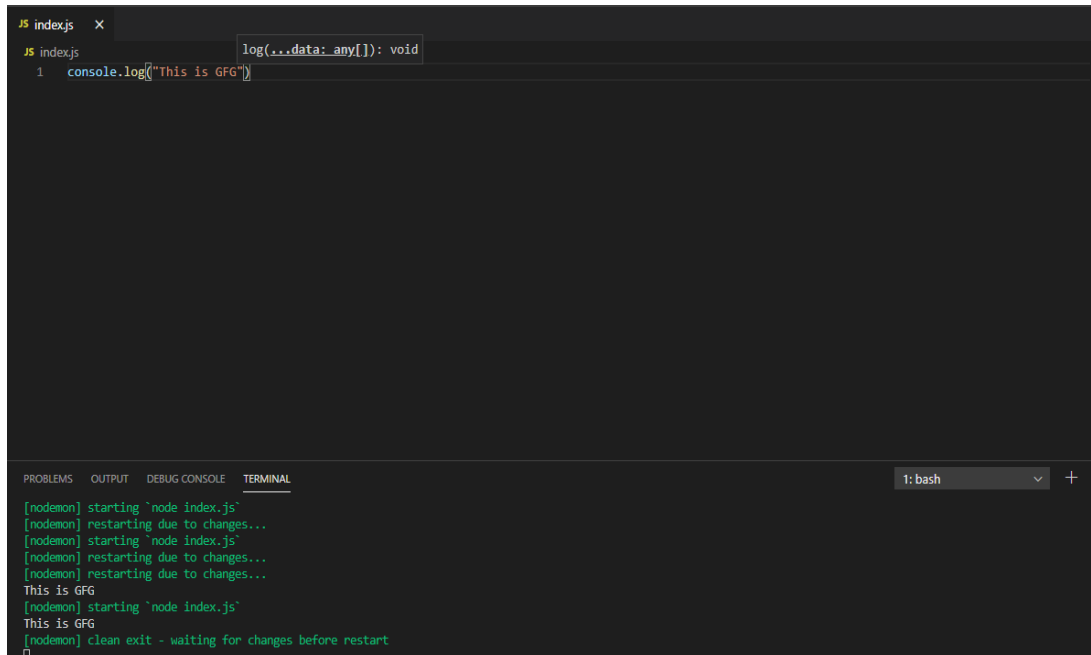
--config file ..... alternate nodemon.json config file to use
-e, --ext ..... extensions to look for, ie. js,pug,hbs.
-x, --exec app ..... execute script with "app", ie. -x "python -v".
-w, --watch path ..... watch directory "path" or files. use once for
                        each directory or file to watch.
-i, --ignore ..... ignore specific files or directories.
-V, --verbose ..... show detail on what is causing restarts.
-- <your args> ..... to tell nodemon stop slurping arguments.

Note: if the script is omitted, nodemon will try to read "main" from
package.json and without a nodemon.json, nodemon will monitor .js, .mjs, .coffee,
.litcoffee, and .json by default.

For advanced nodemon configuration use nodemon.json: nodemon --help config
See also the sample: https://github.com/remy/nodemon/wiki/Sample-nodemon.json
```

**Steps to run the program:** Use the following command to run the file as shown below:

*nodemon index.js*



The screenshot shows a VS Code editor with a file named `index.js` containing the following code:

```
log(...data: any[]): void
1 console.log("This is GFG")
```

Below the editor is a terminal window with the following output:

```
[nodemon] starting 'node index.js'
[nodemon] restarting due to changes...
[nodemon] starting 'node index.js'
[nodemon] restarting due to changes...
[nodemon] restarting due to changes...
This is GFG
[nodemon] starting 'node index.js'
This is GFG
[nodemon] clean exit - waiting for changes before restart
```

It automatically checks the statements and the syntax of the program while writing new statements and show the result on the console.

### ❖ **NPM uuid:**

NPM (Node Package Manager) is a package manager of Node.js packages. There is an NPM package called 'shortid' used to create short non-sequential url-friendly unique ids. Unique ids are created by Cryptographically-strong random values that's why it is very secure. It has support for cross-platform like Node, React Native, Chrome, Safari, Firefox, etc.

### **Command to install:**

*npm install uuid*

### **Syntax to import the package in local file**

```
const {v4 : uuidv4} = require('uuid')
```

### **Syntax to create unique id**

```
const newId = uuidv4()
```

There are some methods defined on shortid modules to create unique ids and customize the ids. some of the methods are illustrates below:

Method	Work
uuid.NIL	The nil UUID string (all zeros)
uuid.parse()	Convert UUID string to array of bytes
uuid.validate()	Test a string to see if it is a valid UUID
uuid.v1()	Create a version 1 (timestamp) UUID
uuid.v3()	Create a version 3 (namespace w/ MD5) UUID
uuid.v4()	Create a version 4 (random) UUID
uuid.v5()	Create a version 5 (namespace w/ SHA-1) UUID
uuid.stringify()	Convert array of bytes to UUID string

#### ❖ The Built-in HTTP Module:

Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

To include the HTTP module, use the **require()** method:

```
var http = require('http');
```

#### ❖ Node.js Path:

The Node.js path module is used to handle and transform files paths. This module can be imported by using the following syntax:

**Syntax:**

```
var path = require("path");
```

### **Body-parser middleware in Node.js:**

Body-parser is the Node.js body parsing middleware. It is responsible for parsing the incoming request bodies in a middleware before you handle it.

### **Installation of body-parser module:**

1. You can install this package by using this command.

```
npm install body-parser
```

2. After installing body-parser you can check your body-parser version in command prompt using the command.

```
npm version body-parser
```

3. After that, you can just create a folder and add a file, for example, index.js. To run this file you need to run the following command.

```
node index.js
```

### **❖ Heroku:**

Heroku is a Platform as a Service (PaaS) product based on AWS, and is vastly different from Elastic Compute Cloud. It's very important to differentiate 'Infrastructure as a Service' and 'Platform as a Service' solutions as we consider deploying and supporting our application using these two solutions.



#### **4. Conclusion**

This web-application is just in its initial stages of development, where we applied the knowledge, we gained about NodeJS and its respective modules. Currently it is a fully functional multi-user video calling web-application, but with minimal features, just the ones that are required for the basic foundation. And in the current circumstances these types of services will remain in the running for quite a long time, both serving formal and informal client-requests. Further development can be done on this to deploy it as a separate website for a group video-calling, commercially.

#### **5. Future Scope**

With the knowledge we have gained during the development of this application, we are confident that in the future, we can make the application more effectively by adding these services.

- Extending this application by providing Authorization service.
- Creating Database and maintaining users.
- Increasing the effectiveness of the application by providing Voice Chat.
- Extending it to Web Support.



## 6. References

- GeeksforGeeks
- StackOverflow