Topic	Video Chat App - SMTP	
Class Description	**Student will learn about SMTP and how to send emails with the Gmail account using nodemailer**	
Class	**C-221**	
Class time	**45 mins**	
Goal	• Learning about SMTP • Sending emails through Gmail using nodemailer	
Resources Required	• Teacher Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Visual Studio Code • Student Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Visual Studio Code	
Class structure	**Warm-Up** **Student - led Activity 1** **Wrap-Up**	**10 mins** **30 mins** **5 mins**
WARM UP SESSION - 10mins		
Teacher Action		**Student Action**
Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today?		**ESR**: Hi, thanks, yes, I am excited about it!

Q&A Session	
Question	**Answer**
In HTML, which tag is used to insert a line break? A. <b> B. C. <a> D. <d>	**B**
Node.js runs on ----------? A. Client end B. Server -end C. Client-server D. None of the above	**B**
STUDENT-LED ACTIVITY - 30mins	
Student Initiates Screen Share	
ACTIVITY • **Understanding about WebRTC and it's functions** • **Fetching the audio and the video for the chat app from the user's browser**	
Teacher Action	**Student Action**
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.	
In the last class, we completed our Video Chat functionality with the help of sockets, WebRTC and PeerJS.	

We also deployed it on Heroku and now it is available for anyone to use. In all the video chat applications, such as Google meet, there is one additional functionality which is quite important - To be able to invite others on our video chat app through email! That's what we are going to do in today's class! We will go over how we can send emails to our friends and family to invite them for a video chat through our application! Are you excited? Let's get started then!	**ESR:** Yes
Do you know which protocol email uses? What is the full form of SMP? That's good! Simple Mail Transfer Protocol is an internet based standard communication protocol for electronic mail transmissions. This protocol is used by all email services out there, including gmail. There are different ports that SMTP can use. Let's understand these ports a bit more - 1. **25 -** It is primarily used for SMTP relaying. In most	**ESR:** SMTP **ESR:** Simple Mail Transfer Protocol

cases, this port ***should be avoided*** as it is blocked by residential IPs and Cloud Hosting Providers. 2. **465 -** IANA (Internet Assigned Number Authorities) has reassigned this port for new services, and it should no longer be used. There are still some old legacy systems that use it. 3. ***587 -*** This is the default mail submission port. It might be busy on a regular server, and if your email service does not work on this port, try 465. Therefore it is safe to say that all modern email services working on SMTP use port 465 to send and retrieve digital emails!	
Now, our server is written in NodeJS, and it offers a special library called ***nodemailer*** that can be used to send emails from NodeJS using SMTP. Remember, that when we deploy an application on Heroku by pushing our code there, we are mostly dependent on both ***package.json*** file and ***yarn.lock*** file to install all the dependencies that we need. Since we are now also needing ***nodemailer*** into our application, let's update these 2 files. You can find the code for ***package.json*** in ***Teacher Activity 2*** ***and*** ***Student Activity 2***. *Teacher helps the student in copying the **package.json** code from Student Activity 2*	*Student copies the code into **package.json** from Student Activity 2*

```
{
  "name": "video-chat-app",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.19.0",
    "ejs": "^3.1.6",
    "express": "^4.17.1",
    "nodemailer": "^6.6.3",
    "peer": "^0.6.1",
    "socket.io": "^4.1.3",
    "uuid": "^8.3.2"
  }
}
```

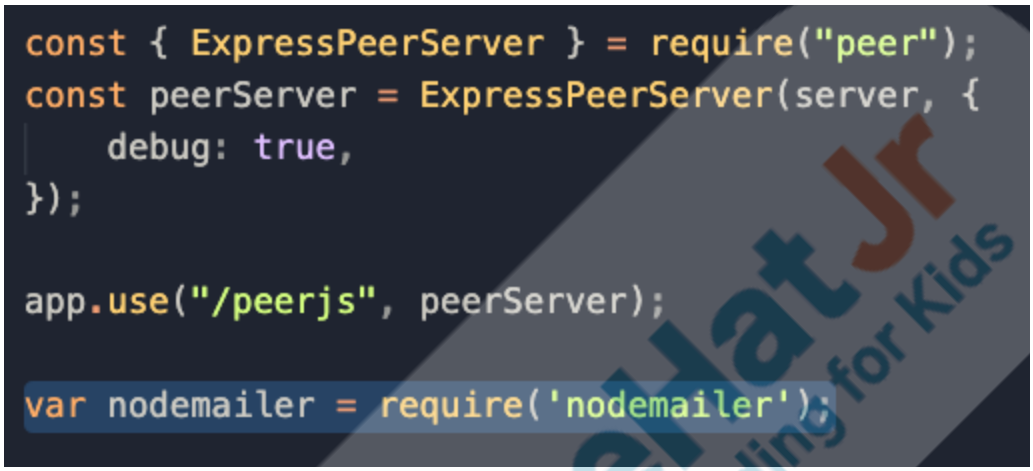
Just like how we have ***package.json*** file, which helps with the ***npm install*** command to know what packages to install, we have ***yarn.lock*** that helps with the ***yarn install*** command to know what exactly needs to be installed.

Both of these commands do the same thing, but ***yarn install*** is much more efficient that ***npm install***, and it is important that we also update the ***yarn.lock*** file. Let's do that -

You can find the code for ***yarn.lock*** in ***Teacher Activity 3*** ***and*** ***Student Activity 3***. *Teacher helps the student in copying the* ***yarn.lock*** *code from Student Activity 3*	*Student copies the code into* ***yarn.lock*** *from Student Activity 3*

```
"@types/component-emitter@^1.2.10":
  version "1.2.10"
  resolved "https://registry.yarnpkg.com/@types/component-emitter/-/component-emitter-1.2.10.tgz#ef
  integrity sha512-bsjleuRKWmGqajMerkzox19aGbscQX5rmmvvXl3wlIp5gMG1HgkiwPxsN5p070fBDKTNSPgojVbuY1+H

"@types/connect@*":
  version "3.4.35"
  resolved "https://registry.yarnpkg.com/@types/connect/-/connect-3.4.35.tgz#5fcf6ae445e4021d1fc221
  integrity sha512-cdeYyv4KWoEgpBISTxWvqYsVy444DOqehiF3fM3ne10AmJ62RSyNkUnxMJXHQWRQQX2eR94m5y1IZyDw
  dependencies:
    "@types/node" "*"

"@types/cookie@^0.4.1":
  version "0.4.1"
  resolved "https://registry.yarnpkg.com/@types/cookie/-/cookie-0.4.1.tgz#bfd02c1f2224567676c154519
  integrity sha512-XW/Aa8APYr6jSVVA1y/DEIZX0/GMKLEVekNG727R8cs56ahETkRAy/3DR7+fJyh7oUgGwNQaRfXCun0+

"@types/cors@^2.8.12", "@types/cors@^2.8.6":
  version "2.8.12"
  resolved "https://registry.yarnpkg.com/@types/cors/-/cors-2.8.12.tgz#6b2c510a7ad7039e98e7b8d3d659
  integrity sha512-vt+kDhq/M2ayberEtJcIN/hxXy1Pk+59g2FV/ZQceeaTyCtCucjL2Q7FXlFjtWn4n15KCr1NE2lNNFhp

"@types/express-serve-static-core@^4.17.18":
  version "4.17.24"
  resolved "https://registry.yarnpkg.com/@types/express-serve-static-core/-/express-serve-static-co
  integrity sha512-3UJuW+Qxhzwjq3xhwXm2onQcFHn76frIYVbTu+kn24LFxI+dEhdfISDFovPB8VpEgW8oQCTpRuCe+0zJ
  dependencies:
    "@types/node" "*"
    "@types/qs" "*"
```

Now, we can be certain that the ***nodemailer*** will be installed on the server when we deploy it to Heroku! Next, let's start writing some code!	

In our ***server.js***, we will first want to ***require()*** nodemailer - *Teacher helps the student in writing the code*	*Student writes the code*
 `const { ExpressPeerServer } = require("peer");` `const peerServer = ExpressPeerServer(server, {` `  debug: true,` `});` `app.use("/peerjs", peerServer);` `var nodemailer = require('nodemailer');`	
With the help of ***nodemailer***, you can create a ***transporter*** that can be used to transport emails on someone's behalf automatically. Let's write the code to create a transporter for our mailer - *Teacher helps the student in writing the code*	*Student opens the file*

```
const transporter = nodemailer.createTransport({
    port: 465,
    host: "smtp.gmail.com",
    auth: {
        user: 'apoorv.goyal@whitehatjr.com',
        pass: '',
    },
    secure: true,
});
```

In this, we have created a ***constant*** called ***transporter***, in which we are calling the ***nodemailer.createTransport()*** function.

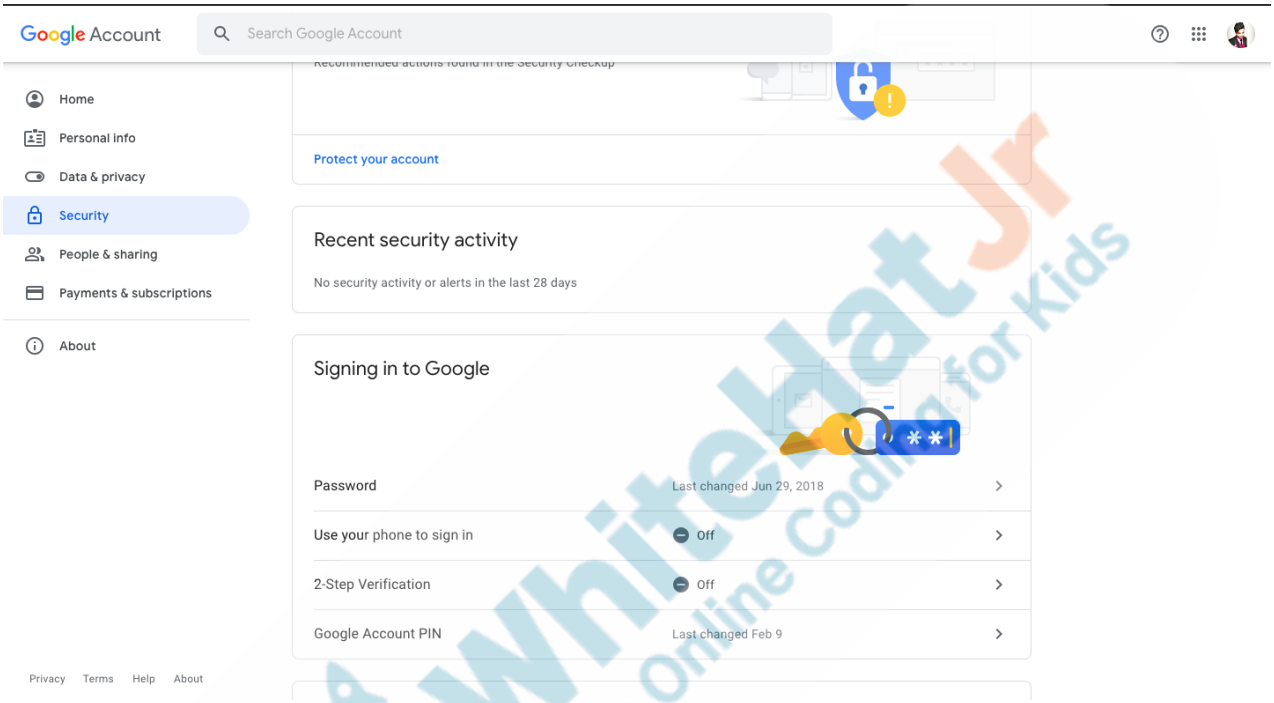
Inside this function, we are passing an object with some data containing -

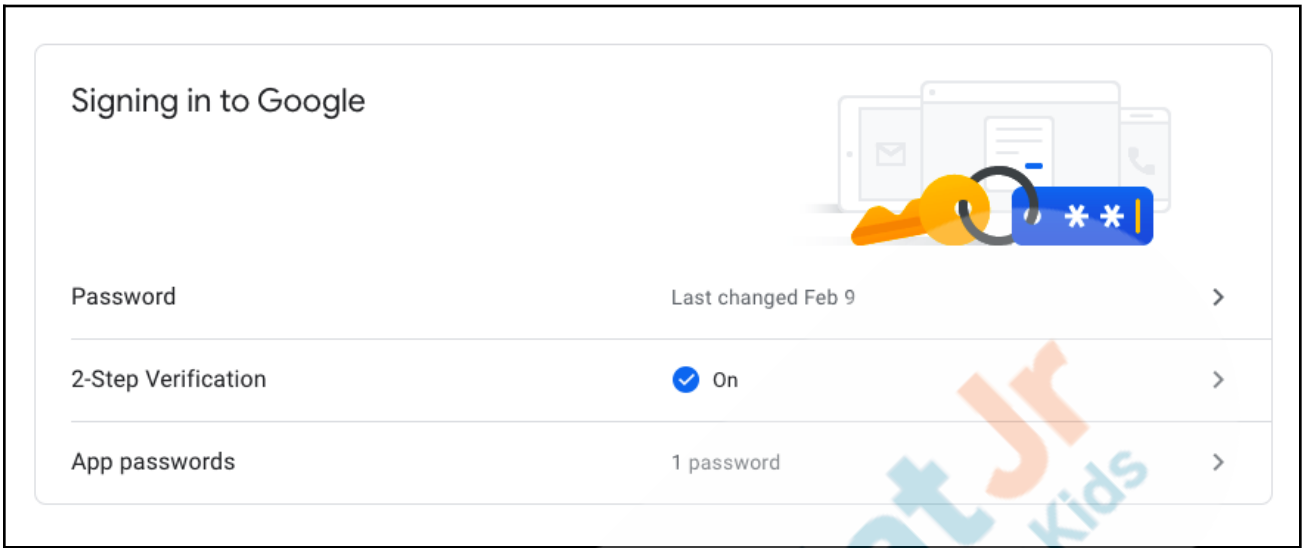
1. ***port*** - Port number that it should use - 465
2. ***host - smtp.gmail.com*** for Gmail's SMTP
3. ***auth -*** That contains a ***user*** and a ***pass*** for email ID and password through which it should send an email
4. ***secure*** - true, to send encrypted emails

Now, we can use our email here, but what about the password?

It will not accept our regular gmail password, due to security reasons, so what google offers is ***app passwords***.

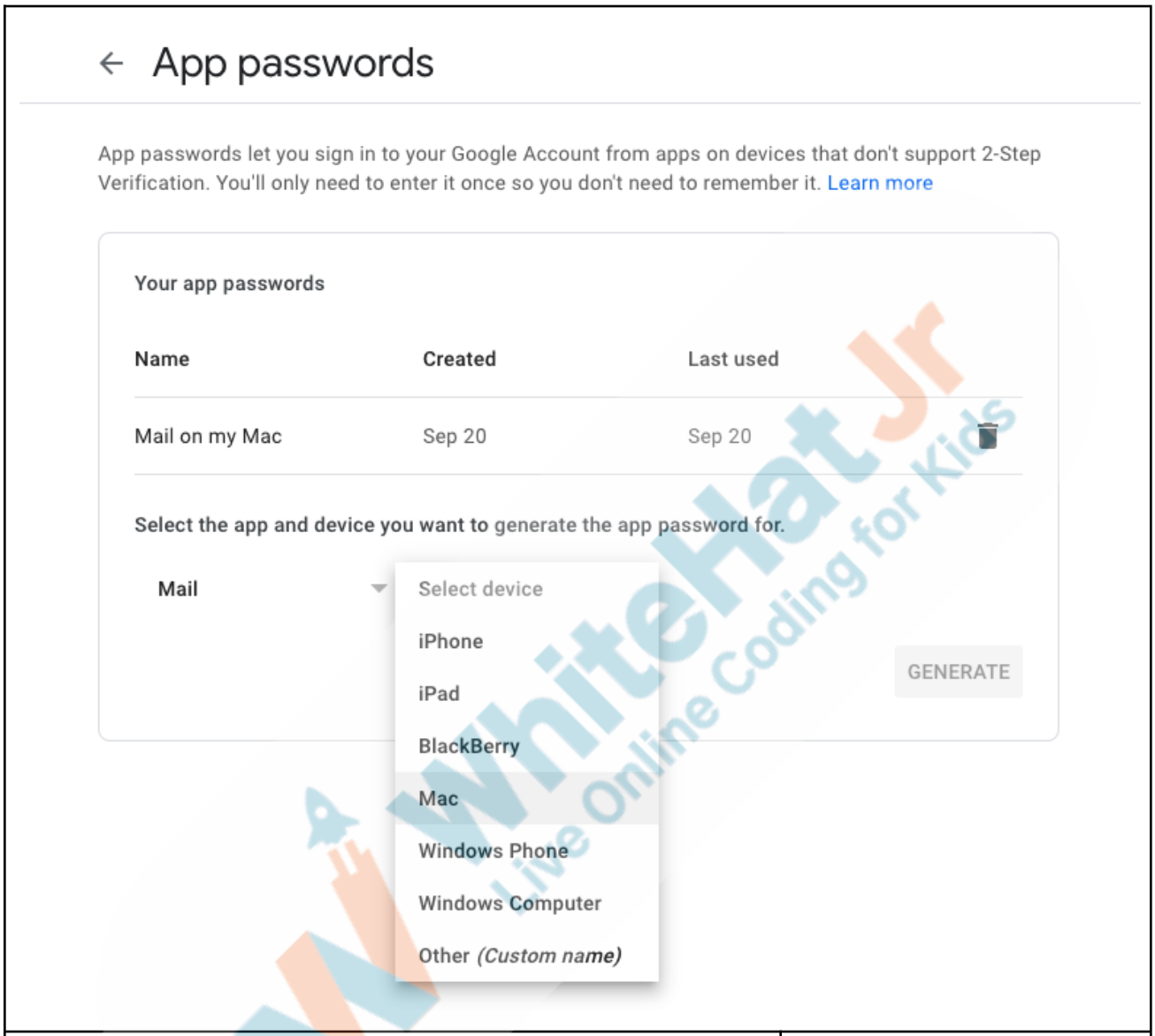
If we are creating an application that is using any of Google's services, google offers this creative password solution with ***app passwords***. It helps us generate passwords that can be used in our apps to access google services, and since ***gmail*** is a google service, we would need to create an app password.

Teacher refers to Teacher Activity 4 and opens it in a new tab	*Student refers to Student Activity 4 and opens it in a new tab*
	
In the ***Signing in to Google*** section, you will need to ensure that your ***2-Step Verification*** is turned on. *Teacher helps the student in following the steps to turn on 2-Step Verification*	*Student turns on 2-Step Verification*
Once done, you will notice that there is another option visible now, called ***App Passwords***. Click on it -	



Click on App Passwords -

Here, select ***Mail*** in the app and Mac (or Windows) for your device *Teacher guides the student*	*Student follows the instructions*



Click on ***Generate*** button and you will see your password -	

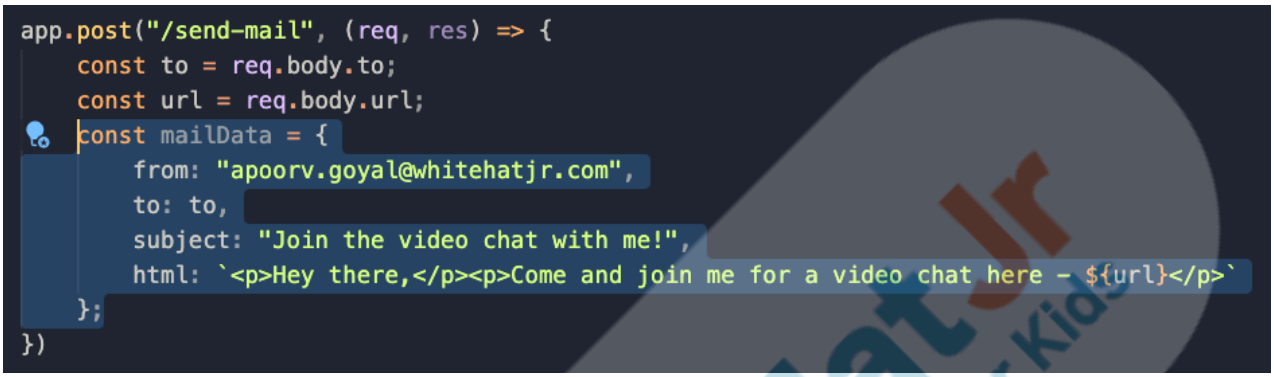
Great! Now, this password can be copied, and pasted into the ***pass*** in our transporter's code so that it can work! *Teacher guides the student in writing the code*	*Student follows the instructions*

```
const transporter = nodemailer.createTransport({
    port: 587,
    host: "smtp.gmail.com",
    auth: {
        user: 'apoorv.goyal@whitehatjr.com',
        pass: 'xsudosfnzmndkvbg',
    },
    secure: true,
});
```

We are almost near! Now, we need to create an API, that can receive a ***POST*** request from the client with the room's ***URL*** and ***To*** (the email id of the person we want to send the invite to) data. *Teacher helps the student in writing the code*	*Student writes the code*

```
app.get("/:room", (req, res) => {
    res.render("index", { roomId: req.params.room });
});

app.post("/send-mail", (req, res) => {
    const to = req.body.to;
    const url = req.body.url;
})
```

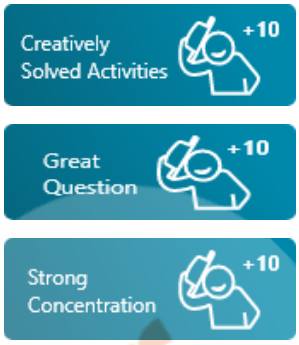
Do note that we are creating a ***post*** request with ***app.post()*** function.	*Student writes the code*

Next, we need to create an object with all the email's data - *Teacher helps the student in writing the code*	
 ``` app.post("/send-mail", (req, res) => { const to = req.body.to; const url = req.body.url; const mailData = { from: "apoorv.goyal@whitehatjr.com", to: to, subject: "Join the video chat with me!", html: `<p>Hey there,</p><p>Come and join me for a video chat here - ${url}</p>` }; }) ```	
Here, in this object, we are setting the ***"from"*** to our email ID, ***"to"*** to the ***email id*** of the receiver, ***subject*** to ***"Join the video chat with me!"***, which would be the subject of our email, and ***"html"*** to the html we want to send in the email. That's right, we can send HTML in the email as well! This HTML also contains the URL that the user needs to click on to join our room! Lastly, we will use the ***sendMail()*** function of the ***transporter*** we created earlier to send the email based on the object ***mailData*** that we created - *Teacher helps the student in writing the code*	*Student writes the code*

```
app.post("/send-mail", (req, res) => {
    const to = req.body.to;
    const url = req.body.url;
    const mailData = {
        from: "apoorv.goyal@whitehatjr.com",
        to: to,
        subject: "Join the video chat with me!",
        html: `<p>Hey there,</p><p>Come and join me for a video chat here - ${url}</p>`
    };
    transporter.sendMail(mailData, (error, info) => {
        if (error) {
            return console.log(error);
        }
        res.status(200).send({ message: "Invitation sent!", message_id: info.messageId });
    });
})
```

Here, we are using the ***transporter.sendMail()*** to send the email with our ***mailData*** as it's first argument, and we have an arrow function to handle success and errors! Inside the function, ***if*** we have an ***error***, then we are logging the error to the console for now, else we are sending a ***response*** with ***res***, whose status code will be ***200***, letting the client know that the Invitation is sent!	
We are halfway through our email invitation functionality! In the next class, we will be completing this app by connecting our API with our client side, and test the functionality too!	
Teacher Guides Student to Stop Screen Share	
WRAP UP SESSION - 5 Mins	
Quiz time - Click on in-class quiz	
Question	**Answer**
Node.js runs on ----------?	**A**

A. Single thread B. Multiple thread C. Double thread D. None of the above	
In SMTP, which of the following commands is used to send the mail address? A. Send_mail B. Post_mail C. Receive_mail D. None of the above	**A**
What does smtp stand for ? A. Simple Mail Protocol B. Sample Mail transfer platform C. Simple Mail Transfer Protocol D. None of the above	**C**
End the quiz panel	
FEEDBACK • **Appreciate the students for their efforts in the class.** • **Ask the student to make notes for the reflection journal along with the code they wrote in today's class.**	
Teacher Action	**Student Action**
You get Hats off for your excellent work!	*Make sure you have given at least 2 Hats Off during the class for:*

In the next class, we will complete our mailer and hence, complete the video chat application!	
Project Discussion In Class 221, we learnt about nodemailer, and how it can be used to send emails through any application. In this project, you'll be creating a mailer to automate reminder emails for a company. You were approached by a small company, who wants to automate their reminder emails about payment. They have been trying to create a mailer and have been struggling really hard. They seek your help! Help them create the emailer!	
Teacher Clicks ✖ End Class	
ADDITIONAL ACTIVITIES	
Additional Activities *Encourage the student to write reflection notes in their reflection journal using markdown.* Use these as guiding questions:	*The student uses the markdown editor to write her/his reflections in the reflection journal.*

• What happened today? ○ Describe what happened. ○ The code I wrote. • How did I feel after the class? • What have I learned about programming and developing games? • What aspects of the class helped me? What did I find difficult?	

ACTIVITY LINKS		
Activity Name	**Description**	**Link**
Teacher Activity 1	Previous Class Code	https://github.com/pro-whitehatjr/PRO-C220-ReferenceCode
Teacher Activity 2	package.json	https://github.com/pro-whitehatjr/PRO-C221-Reference-Code/blob/main/package.json
Teacher Activity 3	yarn.lock	https://github.com/pro-whitehatjr/PRO-C221-Reference-Code/blob/main/yarn.lock
Teacher Activity 4	Google App Passwords	https://myaccount.google.com/security
Teacher Activity 5	Reference Code	https://github.com/pro-whitehatjr/PRO-C221-Reference-Code

Student Activity 1	Previous Class Code	https://github.com/pro-whitehatjr/PRO-C220-ReferenceCode
Student Activity 2	package.json	https://github.com/pro-whitehatjr/PRO-C221-Reference-Code/blob/main/package.json
Student Activity 3	yarn.lock	https://github.com/pro-whitehatjr/PRO-C221-Reference-Code/blob/main/yarn.lock
Student Activity 4	Google App Passwords	https://myaccount.google.com/security