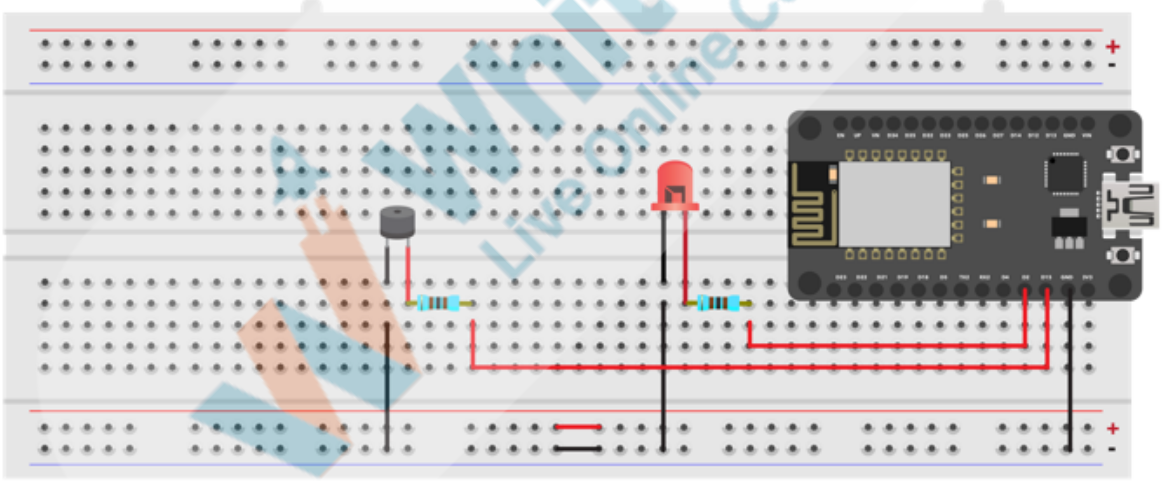
Topic	ESP32 WEB SERVER	
Class Description	**Students will be creating a web server with an ESP32 WIFI Mode that controls outputs (one LED and one buzzer) on the local environment**	
Class	**PRO C247**	
Class time	**50 mins**	
Goal	• Introduction to Wifi Mode • Creation of Web Server • Access to Web Server • Access LED & Buzzer	
Resources Required	• Teacher Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Smartphone • Student Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen	
Class structure	**Warm-Up** **Teacher-Led Activity** **Student-Led Activity** **Wrap-Up**	**10 mins** **20 mins** **20 mins** **05 mins**
Credit & Permissions:	**Code samples used for Firebase-Google Authentication are licensed under the Apache 2.0 License.** **Expo documentation used from - https://expo.io** **Note: Keep this row section only if applicable**	

WARM-UP SESSION - 10 mins	
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? **Following are the WARM-UP session deliverables:** • Greet the student. • Revision of previous class activities. • Quizzes.	**ESR**: Hi, thanks! Yes, I am excited about it! Click on the slide show tab and present the slides
WARM-UP QUIZ Click on In-Class Quiz	
Activity Details **Following are the session deliverables:** • Appreciate the student. • Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.	
TEACHER-LED ACTIVITY-1- 10 mins	
Teacher Initiates Screen Share	
ACTIVITY • **Introduction to Wifi Libraries** • **Creation of Web Server**	
Teacher Action	**Student Action**
Note: The teacher perform below complete activity on her end first to understand the concept of ESP 32 webserver Now, I am sure you are much comfortable with LED's, Buzzers, and sensors?	

Am I right! So let's try something new today! We were discussing so much about IoT, and we learned that in IoT we can control end devices)like a fan, door, LED, and many more on the server, and we even saw a simulation on the Cisco packet tracer too. But don't you want to see real automation on which you actually control your LED & Buzzer on the server just by toggling a button. I know you are excited to do that! The first thing we need to do is to make a local server and we all become pros at creating servers as we have learned so much about networking. Right! But when we create a server in embedded language i.e. C syntaxes/format can be different! Let's learn and create a local server.	**ESR:** Yes! **ESR:** Yes! **ESR:** Yes!
Step -1: Gather the material from the IoT kit: ● 1 x ESP32 ● 1 x USB Cable ● 1 x Breadboard ● 6 x Jumper wires ● 1 x Buzzer ● 1 x LED ● 1	
Step -2: Let's do connections:	

- Supply **negative(GND (-ve))** from the **ESP 32** to the breadboard **negative terminal.**
- Insert **buzzer** and **LED** into the **breadboard**
- Connect the **resistor's one terminal** with a longer leg of the **LED and** the other end of the resistor with ESP32 **GPIO pin numbers D2**
- Connect the second **resistor's one terminal** with a longer leg of the **buzzer.** And the other end of the resistor with ESP32 **GPIO pin numbers D15**
- Connect the short leg of **LED** and **Buzzer** with **GND(0V)** supply.



Step-3 Let's write a code:
The first and most important thing while creating a web server is to include web server libraries.

Using the **WiFi** library, the device will be able to answer an **HTTP** request with your **WiFI credentials.**

After opening a web browser and navigating to your WiFi IP address, the board will respond with HTML content along it will display the input values from the ESP32 board.

include keyword is used to import libraries in embedded language as we used to import in python language

- **WiFi** library: WiFi library will be able to answer all HTTP request
- **WiFiClient**: **WiFiClient** client helps to connect to a specified internet IP address and port.
- **WebServer** library will help to create a web server
- **ESPmDNS** enabled **DNS(Domain Name System) on ESP32**

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include <ESPmDNS.h>
```

After uploading libraries the next step is to connect with ESP32 with the WiFi. For that, we need to use SSID(Wi-Fi credentials i.e WiFi name and WiFi Password)

- **Constant char** is a variable that is used to save WiFi credentials. Set the **SSID** and **password**

Note: Teacher/Student should use their actual WIFi Credentials.

- Load the HTML design string, This string will take

the actual design of the HTML page so use all content of HTML in one string.

- Set **webServer port** number to **80**
- void **handleroot()** function monitors the presence of a webpage request and delivers the requested webpage.
- In response to an accepted request**, server. send** will send a **success** message
- **200** means the request is ok, usually, this will be the standard practice for sending messages for successful web pages.
-

Note" Boilerplate Code ends here

```
const char* ssid = "WR3005N3-757E";
const char* password = "7002949";

String button = "<html><body id='bdy_1' style='height: 100px; width: 100px;'>

WebServer server(80); //http port number

void handleRoot(){
// (192.168.1.1/){
  server.send(200, "text/html", button); //here 200 is Success code
}
```

Note: Teacher starts writing code from here

In case the HTTP request fails, **the handleNotFound()** function comes into play.

- **string message** is a variable along with datatype string which will save the message **"File Not Found".**
- **/n** represents new line

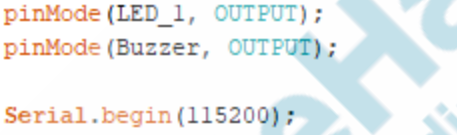
- **192,178.0.1** will be the local address to access the server.
- The server will try to make the success request using **get()** and **post()** method
- **404** means the requested page could not be found but may be available again in the future.

```
void handleNotFound() {

  String message = "File Not Found\n\n";
  message += "URI: "; // (192.168.1.1/page1/on)
  message += server.uri();
  message += "\nMethod: ";
  message += (server.method() == HTTP_GET) ? "GET" : "POST";
  message += "\nArguments: ";
  message += server.args();
  message += "\n";
  for (uint8_t i = 0; i < server.args(); i++) {
    message += " " + server.argName(i) + ": " + server.arg(i) + "\n";
  }
  server.send(404, "text/plain", message);

}
```

As we have set up a server, now the next thing is to define the GPIO pins

- define GPIO pin along with data type **int** for **LED_1 D2**
- define GPIO pin along with data type **int** for **Buzzer D15**

```
const int  LED_1 = 2;
const int  Buzzer = 15;
```

Initialize using **void setup()** function • **PinMode()** configures the specified pin to behave either as an input or an output. As we want to act this LED & Buzzer pin as output we will use **OUTPUT** here. • **Serial. begin(115200)** is used to measure the speed of data exchange. This tells the Arduino to get ready to exchange messages with the Serial Monitor at a data rate of 9600 bits per second. That's 9600 binary ones or zeros per second and is commonly called a baud rate.	
	
Currently, we have set up the webserver but still, we need to write the code to make LED and Buzzer work.	
Teacher Stops Screen Share	
So now it's your turn. Please share your screen with me.	
Can you write the algorithm to make your Buzzer and LED on? Let's try. I will guide you through it.	
STUDENT-LED ACTIVITY-1 - 20 mins	
• Ask the student to press the ESC key to come back to the panel. • Guide the student to start Screen Share. • The teacher gets into Full Screen.	
Student Initiates Screen Share	

<table>
<tr><td colspan="2">ACTIVITY

• Student Activity description (in bullet points).</td></tr>
<tr><td>Teacher Action</td><td>Student Action</td></tr>
<tr><td>Note: Guide the student to collect the material from the IoT Kit.</td><td></td></tr>
<tr><td>Step -1:Gather the material from the IoT kit:

• 1 x ESP32
• 1 x USB Cable
• 1 x Breadboard
• 4 x Jumper wires
• 1 x Potentiometer
• 1 x Rotary Potentiometer</td><td></td></tr>
<tr><td>Step -2: Let’s do connections:

• Supply positive (VCC (+ve)) from the ESP 32 to the breadboard positive terminal.
• Supply negative(GND (-ve)) from the ESP 32 to breadboard negative terminal
• Take the DHT11 sensor (female jumper wires are already connected with DHT11)
• Take three male jumper wires to insert into DHT11 female jumper wires.
• Connect VCC (+ve) of DHT11 with VCC (+ve) of the breadboard
• Connect GND(-ve) of DHT11 with GND(-ve) of the breadboard
• Connect data/output pin of DHT11 with D15 of the ESP32</td><td></td></tr>
<tr><td>Step-3 Let’s write a code:

Define Pins</td><td></td></tr>
</table>

- define **DHTPIN 15**
- define **DHTPIN DHT11**

```
if (MDNS.begin("esp32")) { //esp32.local/
  Serial.println("MDNS responder started");
```

Initialize the setup()

- **Serial. begin(9600)** is used for data exchange speed. This tells the Arduino to get ready to exchange messages with the Serial Monitor at a data rate of 9600 bits per second. That's 9600 binary ones or zeros per second and is commonly called a baud rate.
- **Serial.println is** used to print data. Print ("**DHT11 sensor!**")
- **dht.begin()** is used to begin the process

```
server.on("/", handleRoot); // esp32.local/

server.on("/roomLight/on", []() {
  server.send(200, "text/html",button);
  digitalWrite(LED_1,HIGH);
  //delay(200);

});

server.on("/roomLight/off", []() {
  server.send(200, "text/html",button);
  digitalWrite(LED_1,LOW);
});
```

To execute the main process write the **void loop()**

- **server.on** will on the room bell i.e Buzzer
- **server.send** send success message on web page
- As soon as request message got success, It will make the Buzzer **High** using **digitalWrite()** function
- Repeat the same process to make Buzzer **LOW using digitalWrite()**
- If there is any error then the **handlenotfound()**

function will call.

```
server.on("/roomBell/on", []() {
  server.send(200, "text/html",button);
  digitalWrite(Buzzer,HIGH);
});

server.on("/roomBell/off", []() {
  server.send(200, "text/html",button);
  digitalWrite(Buzzer,LOW);
});

server.onNotFound(handleNotFound);
```

Now, it's time to write the code for the LED

- **server.begin()** will start the server.
- **Serial.println** is used to print **("HTTP server started")**
- Make the **LED_1 LOW/HIGH** using **digitalWrite()** function

```
server.begin();
Serial.println("HTTP server started");

digitalWrite(LED_1,LOW);
digitalWrite(Buzzer,LOW);
```

Call the main function

- Call the main function **server.handleClient()**

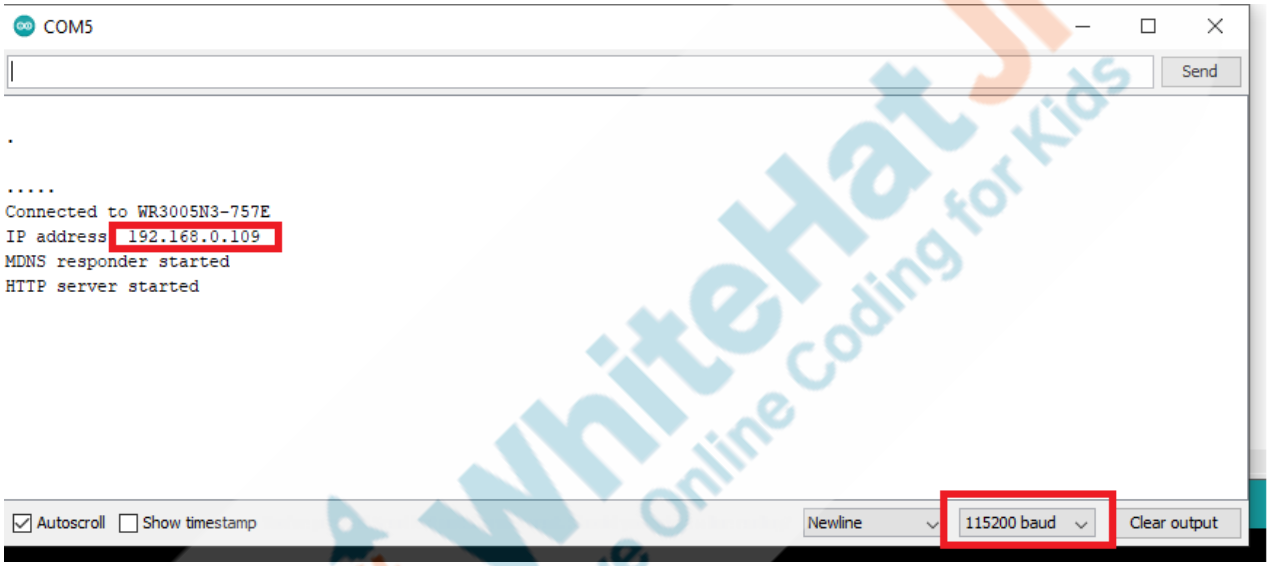
```
void loop(void) {
  server.handleClient();
}
```

Output:
Compile and upload the program to ESP32 board using Arduino IDE

- **Verify** the program by clicking the tick option
- **Upload** the program by clicking the arrow option

Note: If the port is not selected, insert the USB cable in Computer's port and select the port

- Go to **Tools** and select **Serial Monitor**
- Select **baud rate** at **115200** as shown in screenshot
- Reset the **ESP32** by pressing the **EN** button. The ESP32 connects to **Wi-Fi** and displays its IP address on the Serial Monitor. Copy that **IP address**, open the browser, paste the ESP32 **IP address**



- Copy the address and paste it on the browser
- Following window will open
- Control your **LED** & **Buzzer** On and Off using a web server.

Great! So we developed our own web server.

Isn't fun!

So we learned about servers and how to control end devices (like LED, Buzzer) on local servers.

Teacher Guides Student to Stop Screen Share

WRAP-UP SESSION - 05 mins

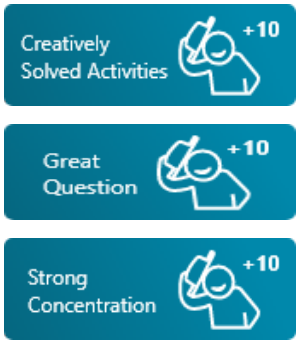
Teacher Starts Slideshow
Slide 14-18

Activity details

Following are the WRAP-UP session deliverables:

- Appreciate the student.
- Revise the current class activities.
- Discuss the quizzes.

WRAP-UP QUIZ

Click on In-Class Quiz	
Activity Details **Following are the session deliverables:** • Explain the facts and trivia • Next class challenge • Project for the day • Additional Activity (Optional)	
FEEDBACK • **Appreciate and compliment the student for trying to learn a difficult concept.** • **Get to know how they are feeling after the session.** • **Review and check their understanding.**	
Teacher Action	**Student Action**
You get “hats-off” for your excellent work! In the next class, we will learn about IoT platforms	*Make sure you have given at least 2 hats-off during the class for* 
PROJECT OVERVIEW DISCUSSION Refer the document below in Activity Links Sections	
Teacher Clicks ✖ End Class	

ACTIVITY LINKS		
Activity Name	**Description**	**Links**
Teacher Activity 1	Boilerplate Code	https://github.com/procodingclass/PRO-C247-Teacher-Boilerplate
Teacher Activity 2	Reference Code	https://github.com/procodingclass/PRO-C247-Reference-Code
Teacher Reference 1	In-Class Quiz	https://s3-whjr-curriculum-uploads.whjr.online/ada33ff9-cef8-4a7f-9d8b-3acf3fa7d9f9.docx
Student Activity 1	Boilerplate Code	https://github.com/procodingclass/PRO-C247-Student-Boilerplate-Code