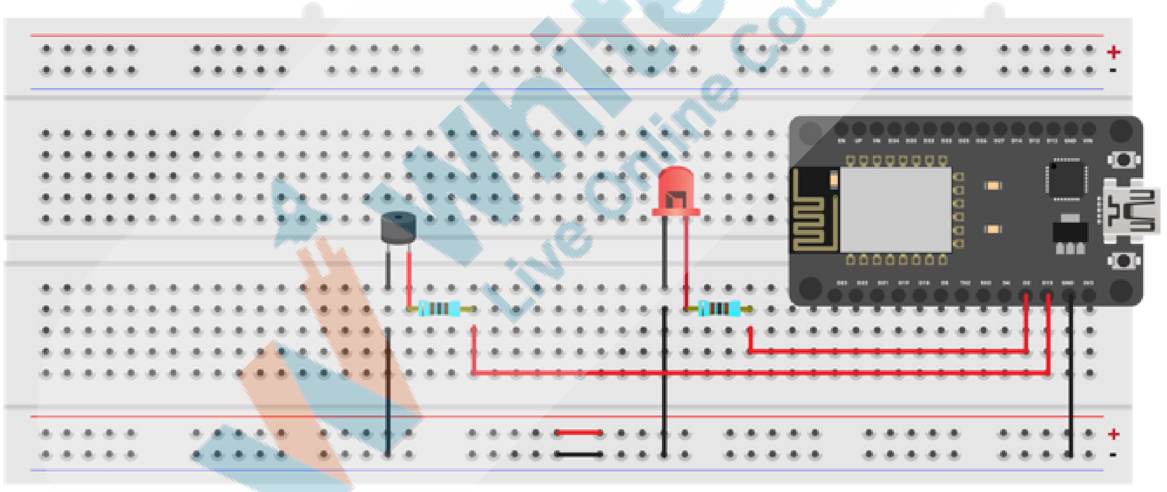
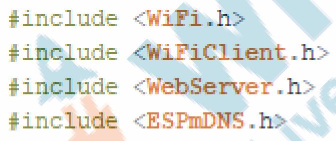
WhiteHat Jr
Live online Coding for Kids

| 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<br>● Creation of Web Server<br>● Access to Web Server<br>● Access LED & Buzzer | |
| Resources Required | ● Teacher Resources:<br>○ Laptop with internet connectivity<br>○ Earphones with mic<br>○ Notebook and pen<br>○ Smartphone<br><br>● Student Resources:<br>○ Laptop with internet connectivity<br>○ Earphones with mic<br>○ Notebook and pen | |
| Class structure | **Warm-Up**<br>**Teacher-Led Activity**<br>**Student-Led Activity**<br>**Wrap-Up** | **10 mins**<br>**20 mins**<br>**20 mins**<br>**05 mins** |
| Credit & Permissions: | **Code samples used for Firebase-Google Authentication are licensed under the [Apache 2.0 License](#).**<br>**Expo documentation used from - [https://expo.io](https://expo.io)**<br>**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?<br><br>**Following are the WARM-UP session deliverables:**<br>● Greet the student.<br>● Revision of previous class activities.<br>● Quizzes. | **ESR**: Hi, thanks!<br>Yes, I am excited about it!<br><br>Click on the slide show tab and present the slides |
| **WARM-UP QUIZ**<br>Click on In-Class Quiz | |
| **Activity Details**<br><br>**Following are the session deliverables:**<br>● Appreciate the student.<br>● Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students. | |
| **TEACHER-LED ACTIVITY - 10 mins** | |
| **Teacher Initiates Screen Share** | |
| **<u>ACTIVITY</u>**<br><br>● **Introduction to Wifi Libraries**<br>● **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*<br><br>Now, I am sure you are much comfortable with LED's, Buzzers, and sensors? | |

| | |
|---|---|
| Am I right!<br><br>So let's try something new today!<br><br>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.<br><br>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!<br><br>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.<br><br>Right!<br><br>But when we create a server in embedded language i.e. C syntaxes/format can be different!<br><br>Let's learn and create a local server. | **ESR:** Yes!<br><br>**ESR:** Yes!<br><br>**ESR:** Yes! |
| **Step -1:Gather the material from the IoT kit:**<br><br>● 1 x ESP32<br>● 1 x USB Cable<br>● 1 x Breadboard<br>● 6 x Jumper wires<br>● 1 x Buzzer<br>● 1 x LED | |
| **Step -2: Let's do connections:** | |

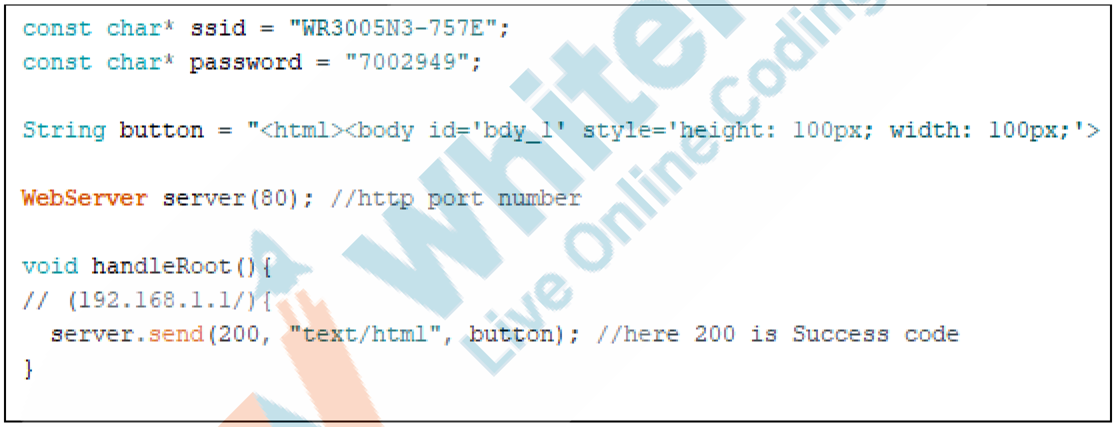
| | |
|---|---|
| • Supply **negative(GND (-ve))** from the **ESP 32** to the breadboard **negative terminal.**<br>• Insert **buzzer** and **LED** into the **breadboard**<br>• Connect the **resistor's one terminal** with a longer leg of the **LED and** the other end of the resistor with ESP32 **GPIO pin numbers 15**<br>• 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 15**<br>• Connect the short leg of **LED** and **Buzzer** with **GND(0V)** supply. | |
|  | |
| **Step-3 Let's write a code:**<br>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.**<br><br>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.<br><br>**include keyword**  is used to import libraries in embedded language as we used to import in python language<br><br>● **WiFi** library: WiFi library will be able to answer all HTTP request<br><br>● **WiFiClient**:  **WiFiClient** client helps to connect to a specified internet IP address and port.<br><br>● **WebServer** library will help to create a  web server<br><br>● **ESPmDNS** enabled **DNS(Domain Name System) on ESP32** | |
| <br>`#include <WiFi.h>`<br>`#include <WiFiClient.h>`<br>`#include <WebServer.h>`<br>`#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)<br><br>● **Constant char** is a variable that is used to save WiFi credentials. Set the **SSID** and **password**<br><br>*Note: Teacher/Student should use their actual WIFi Credentials.*<br><br>● 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*

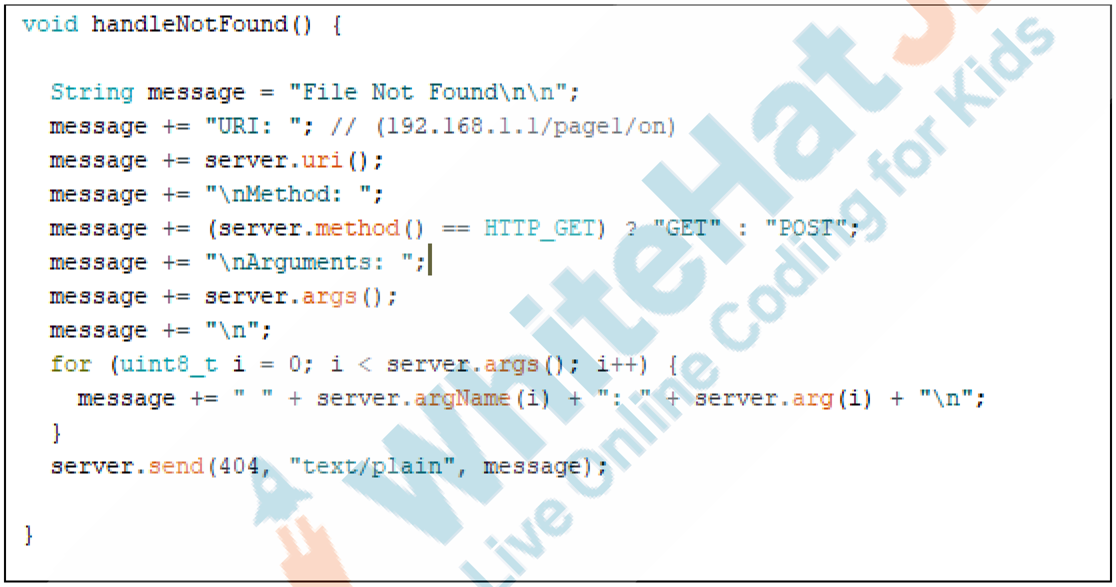
In case the HTTP request fails, **the handleNotFound()** function comes into play.

- **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

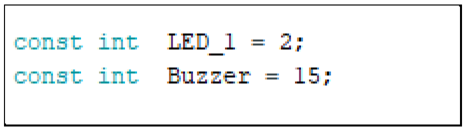
- **404** means the requested page could not be found but may be available again in the future.



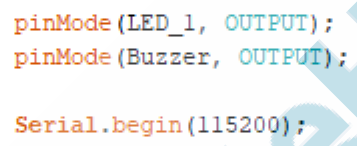
```
void handleNotFound() {

  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**

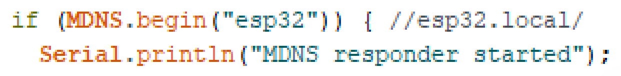


```
const int  LED_1 = 2;
const int  Buzzer = 15;
```

| | |
|---|---|
| Initialize using **void setup()** function<br><br>● **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.<br><br>● **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. | |
| <br>`pinMode(LED_1, OUTPUT);`<br>`pinMode(Buzzer, OUTPUT);`<br><br>`Serial.begin(115200);` | |
| 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.<br>Please share your screen with me. | |
| Can you write the algorithm to make your Buzzer and LED on?<br>Let's try. I will guide you through it. | |
| STUDENT-LED ACTIVITY - 20 mins | |
| ● Ask the student to press the ESC key to come back to the panel.<br>● Guide the student to start Screen Share.<br>● The teacher gets into Full Screen. | |
| Student Initiates Screen Share | |

| ACTIVITY<br>• Student Activity description (in bullet points). | |
|---|---|
| Teacher Action | Student Action |
| *Note: Guide the student to collect the material from the IoT Kit.* | |
| **Step -1:Gather the material from the IoT kit:**<br>• 1 x ESP32<br>• 1 x USB Cable<br>• 1 x Breadboard<br>• 4 x Jumper wires<br>• 1 x Potentiometer<br>• 1 x Rotary Potentiometer | |
| **Step -2: Let's do connections:**<br>• Supply **positive (VCC (+ve))** from the ESP 32 to the breadboard positive terminal.<br>• Supply **negative(GND (-ve))** from the ESP 32 to breadboard negative terminal<br>• Take the **DHT11** sensor (female jumper wires are already connected with DHT11)<br>• Take three male jumper wires to insert into **DHT11** female jumper wires.<br>• Connect **VCC (+ve)** of **DHT11** with **VCC (+ve)** of the breadboard<br>• Connect **GND(-ve)** of **DHT11** with **GND(-ve)** of the breadboard<br>• Connect **data/output pin** of **DHT11** with **D15** of the **ESP32** | |
| **Step-3 Let's write a code:**<br><br>**Define Pins** | |

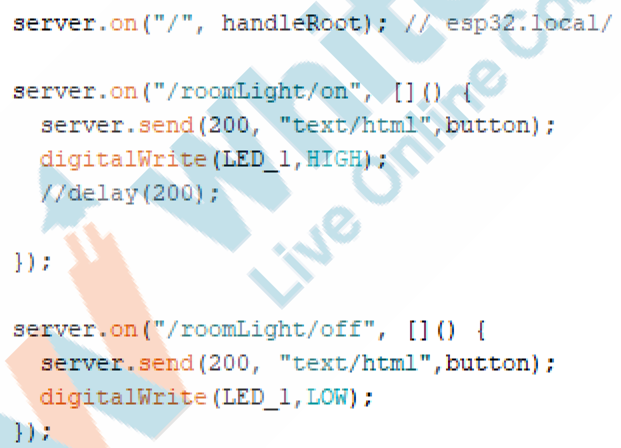
- define **DHTPIN  15**
- define **DHTPIN  DHT11**



```
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()**

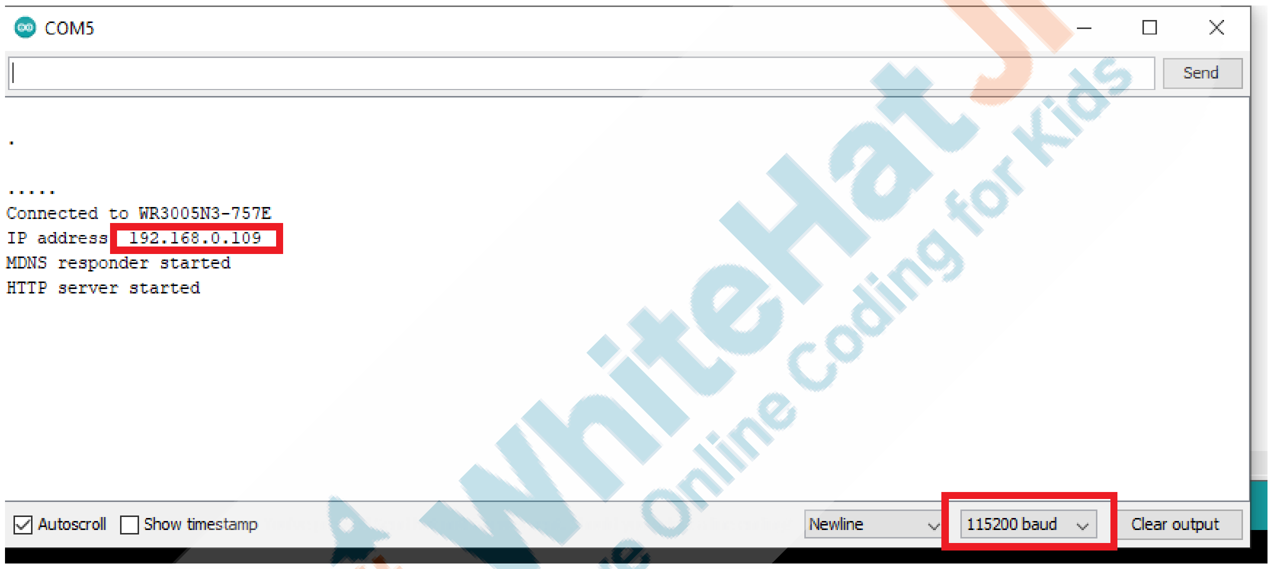
```
void loop(void) {
  server.handleClient();
}
```

**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

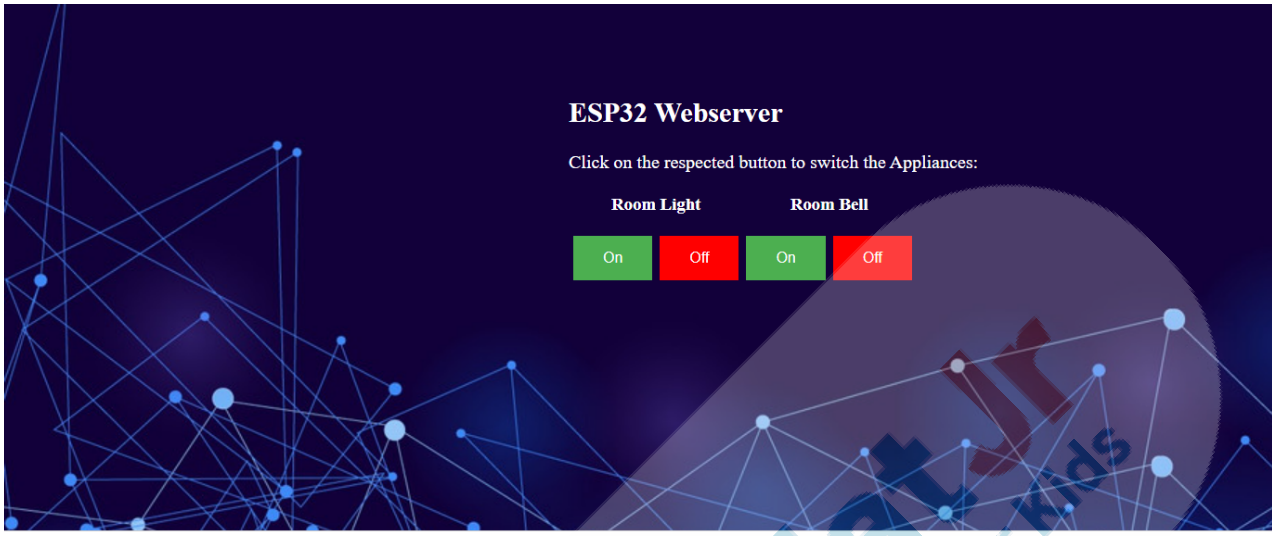
WhiteHat Jr
Live online Coding for Kids

| | |
|---|---|
| *Note: If the port is not selected, insert the USB cable in Computer's port and select the port*<br>• Go to **Tools** and select **Serial Monitor**<br>• Select **baud rate** at **115200** as shown in screenshot<br>• 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** | |



```
.

.....
Connected to WR3005N3-757E
IP address 192.168.0.109
MDNS responder started
HTTP server started
```

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

| |
|---|
|   ESP32 Webserver<br>Click on the respected button to switch the Appliances:<br>Room Light<br>Room Bell<br>On<br>Off<br>On<br>Off |
| Great! So we developed our own web server.<br><br>Isn't fun!<br><br>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**<br>**Slide 14-18** |
| **Activity details**<br><br>**Following are the WRAP-UP session deliverables:**<br>● Appreciate the student.<br>● Revise the current class activities.<br>● Discuss the quizzes. |
| **WRAP-UP QUIZ** |

Click on In-Class Quiz

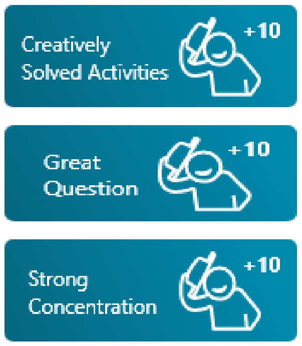
**Activity Details**

**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!<br><br>In the next class,we will learn about IoT platform | *Make sure you have given at least 2 hats-off during the class for:*<br> Creatively Solved Activities +10<br>Great Question +10<br>Strong Concentration +10 |

**PROJECT OVERVIEW DISCUSSION**

Refer the document below in Activity Links Sections

**Teacher Clicks** ✖ End Class

| **ADDITIONAL ACTIVITIES** (Optional) | |
|---|---|
| **Additional Activities** | |

| **ACTIVITY LINKS** | | |
|---|---|---|
| **Activity Name** | **Description** | **Links** |
| Teacher Activity 2 | Boilerplate Code | https://github.com/procodingclass/PRO-C247-Teacher-Boilerplate |
| Teacher Activity 3 | Reference Code | https://github.com/procodingclass/PRO-C247-Reference-Code |
| Teacher Reference 1 | Project | https://s3-whjr-curriculum-uploads.whjr.online/00f7ea0d-afbc-4925-82b4-819c30d1ab3f.docx |
| Teacher Reference 2 | Project Solution | https://github.com/procodingclass/PRO-C247-Project-Solution |
| Teacher Reference 4 | 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 |