1. Day 1 – Web Server
   1. Prepare Arduino for use with the ESP32
   2. as Station
   3. as Soft AP
2. Day 2 – Accessing other Web Pages, Formatting Web Pages, and Git Hub
   1. HTML
   2. CSS
   3. GitHub
   4. Send Email
3. Day 3 – Pull Web Page Data, DHT22, Database
   1. Access other web pages. Pull data.
   2. Write to Google Sheet!!! Store data. Build table and view from web page.
4. Day 4 – Bluetooth and OTA Updates.

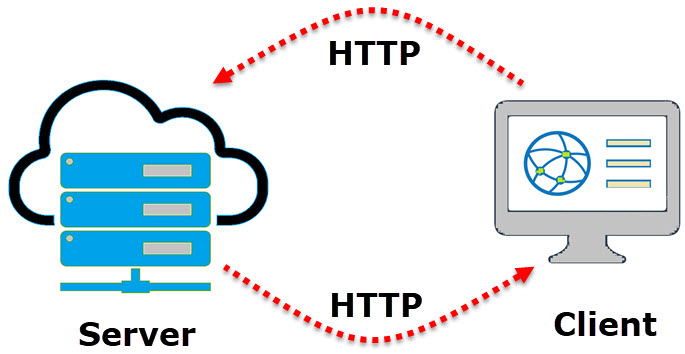
Blynk App??:

* 1. <https://www.techiesms.com/uploading-code-on-esp32-over-wi-fi-basicota/>
  2. <https://www.techiesms.com/uploading-code-on-esp32-over-wi-fi-web-updater-ota/>
  3. Bluetooth may “coexist” with WIFI using the “coexistance module”.

1. Day 5 – Put it all together
   1. Use the DHT22 to read Temperature and Humidity data.
   2. Write this data to a Google Sheet.
   3. In the Google Sheet, create a graph and find the URL for that graph.
   4. Use the ESP32 to serve a webpage with the URL of the graph.

Day 1 – Web Server

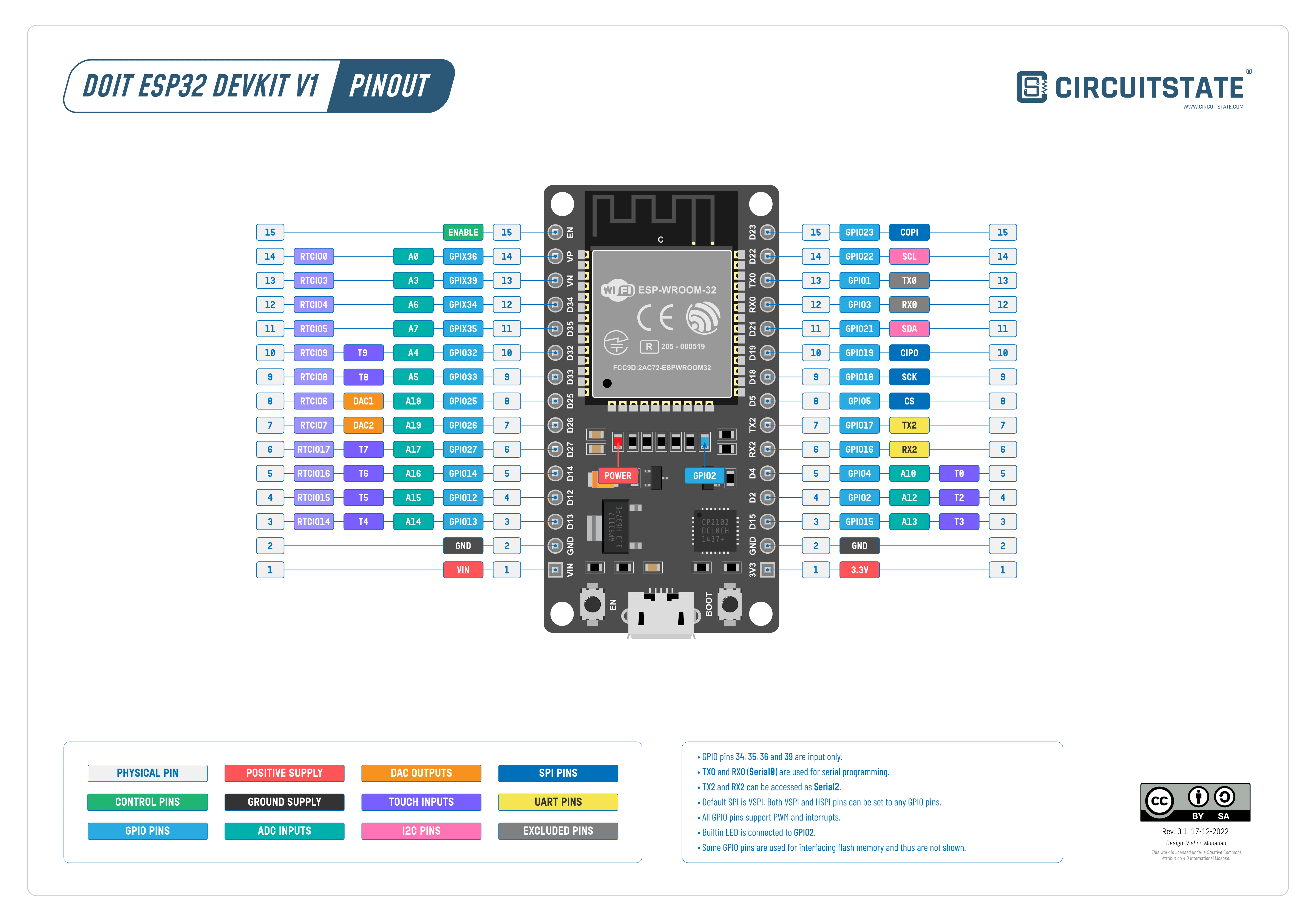
1. A **web server** = combination of hardware and software responsible for maintaining, fetching, and serving web pages to Clients.
2. **Client** = a user or software that makes a request from a webpage.
   1. A Web Client makes a “Request”. The Web Server delivers a “Web Page” to the Client.
   2. The Web Server continues “listening” for a request.
      1. If the requested page is not found, the server responds with an
         1. “HTTP 404 Error”, “Page not found.”



1. **Web Page(Site)** = single hypertext document (transmitted as HTML) on the World Wide Web.
   1. This may include hyperlinks to or from other webpages.
2. **Web Browse**r = software that renders and displays a web page.

Our business this week will be to serve up web pages from the ESP32 microcontrol board.

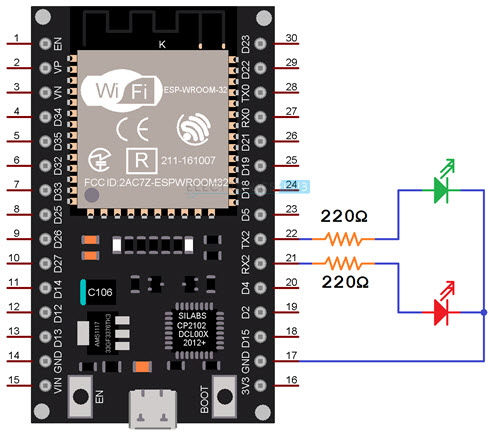
ESP32 Documentation from Espressif: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/>



1. To build webpages, the HTML language is employed.
   1. **HTML** = Hypertext Markup Language; It defines the meaning and structure of web content.
      1. “**Hypertext**” refers to the ability to link other webpages via text URLs.
      2. **URL** = Uniform (Universal) Resource Locator; the address of a resource on the internet that consists of the communication protocol (HTTP or HTTPS) followed by the name or address of a computer on the network, often also including locating information such as directory and file names.
2. The ESP32 Wi-Fi Module operates in 3 Operating Modes:
   1. Station Mode (STA)
      1. Module connects to existing WiFi Network, setup by a Wireless Router.
      2. The ESP32 Wi-Fi Module connects to an existing Wi-Fi Network or Router using the router’s SSID and Password. The Router then assigns a local IP Address for the ESP32.
         1. **IP** = Internet Protocol; the set of rules for routing and addressing packets of data for travel across networks to ensure that they arrive properly at their destination. Data traversing the internet is partitioned into pieces called “**Packets**”.
      3. Our Mobile Phones and Laptops connect using Station Mode.
   2. Soft Access Point Mode (AP)
      1. The ESP32 Module creates its own WiFi Network, like a wireless Router, so that other stations like Mobile Phones, other ESP32s, etc. Can connect to that network.
      2. In Soft Access Point Mode (AP), the ESP32 does not have a Wired Ethernet connection to the internet.
      3. While configuring the ESP32 in AP Mode, set the SSID and PW for the network so that other devices can connect to that network using the credentials chosen.
   3. Station + Soft AP Mode
      1. Combination of Station Mode and Soft AP Mode. The ESP32 acts as both the Station (connecting to a Wifi Network) and an Access Point (serving internet).

Try it!

1. Add the ESP32 to the list of recognized Arduino Boards:
   1. File -> Preferences: “Additional Board Manager URLs”
      1. <https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json>
      2. Note, if you require additional URLs, separate each with commas.
   2. Tools -> Boards -> Boards Manager.
      1. Search for the “ESP32” board and install the “ESP32 by Espressif Systems”.
   3. Tools -> Board: “DOIT ESP32 DEVKIT V1”
   4. If you are unable to interface with the Arduino over Serial Communiation, then install:
      1. <https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers?tab=downloads>
      2. Or follow: <https://sparks.gogo.co.nz/ch340.html> for the CH340 Driver
2. Download or copy/paste Code from Git Hub: <https://github.com/antmhernandez/ESP32-Basics>
   1. Load code to the ESP32 (separately) for both: (<https://www.electronicshub.org/esp32-web-server/>)
   2. Begin with Soft AP Mode, then try Station Mode.



### **Soft Access Point Mode**

* <https://randomnerdtutorials.com/esp32-access-point-ap-web-server/>

### **Station Mode**

* Wire the ESP32 as follows: Pin 21 to red LED +, Pin 22 to green LED +, both to GND.
* <https://gist.github.com/elktros/a39d167e55625396ad6df57b89b00ca7#file-esp32-web-server-ino>
* Change SSID and PW.

### **Email SMTP and SMS Messaging:**

1. SMTP Email: <https://randomnerdtutorials.com/esp32-send-email-smtp-server-arduino-ide/>
2. **SMTP** = Simple Mail Transfer Protocol; internet standard for email transmission.
3. Download the “ESP-Mail-Client" Library.
   1. This allows the ESP32 Module to send and receive emails with or without attachments via SMTP and IMAP servers.
   2. Install the ESP-Mail-Client:
      1. Sketch -> Include Library -> Manage Libraries. Search for “ESP Mail Client” by Mobizt.
4. Create ESP32 Email Account:
   * 1. **Do not use your main personal email to send emails via ESP32**. If something goes wrong in your code or if by mistake you make too many requests, you can be banned or have your account temporarily disabled
     2. **Use “incognito” view to create a new profile without logging out of yours.**
5. Create an App Password
   1. An **app password** =is a password that is generated so that a third-party app can connect to your Gmail account. You then use it instead of your Google account password when using that app.
   2. If using Gmail, in the Navigation Panel, select Security.
      1. Under “Signing in to Google”, select 2-Step Verification -> Get Started
         1. Complete the 2-step process.
      2. Navigate to <https://myaccount.google.com/apppasswords>
         1. Mail. “Custom Name” (Ex. “ESP32”)
         2. Save the generated App Password for reference!
6. Note the settings for your mail client’s SMTP server:
   1. Gmail SMTP Server Settings
      1. SMTP Server: smtp.gmail.com
      2. SMTP username: Complete Gmail address
      3. SMTP password: Your Gmail password
      4. SMTP port (TLS): 587
      5. SMTP port (SSL): 465
      6. SMTP TLS/SSL required: yes
   2. Outlook SMTP Server Settings
      1. SMTP Server: smtp.office365.com
      2. SMTP Username: Complete Outlook email address
      3. SMTP Password: Your Outlook password
      4. SMTP Port: 587
      5. SMTP TLS/SSL Required: Yes
   3. Live or Hotmail SMTP Server Settings
      1. SMTP Server: smtp.live.com
      2. SMTP Username: Complete Live/Hotmail email address
      3. SMTP Password: Your Windows Live Hotmail password
      4. SMTP Port: 587
      5. SMTP TLS/SSL Required: Yes
   4. If you’re using another email provider, you need to search for its SMTP Server settings.
      1. First, search for “Mail Provider” + “App Password” for instructions.
      2. Then, search for your “Mail Provider” + “SMTP Server Settings”
7. Email HTML or Raw Text: <https://github.com/antmhernandez/ESP32-Basics/blob/main/SMTP-Raw-Text-or-HTML-Mail.ino>
8. SMS Text requires Twilio or other paid service: <https://techtutorialsx.com/2021/12/01/esp32-sending-an-sms-with-twilio/>

Day 2 – Accessing other Web Pages, Formatting Web Pages, and Git Hub

1. **Read through Day 1 Code: Web Server Soft AP and Web Server Station.**
   1. **Comprehend each piece with Walkthrough.**
2. HTML:
   1. HTML is the language that is rendered to display a website. It is a formatting and text-output language.
   2. <https://www.w3schools.com/html/default.asp>
3. CSS:
   1. With CSS, you can control the color, font, the size of text, the spacing between elements, how elements are positioned and laid out, what background images or background colors are to be used, different displays for different devices and screen sizes, and much more
   2. <https://www.w3schools.com/html/html_css.asp>
4. Example of CSS in HTML for ESP32 Client Webpage:

String SendHTML(uint8\_t led1stat,uint8\_t led2stat){  
 String ptr = "<!DOCTYPE html> <html>\n";  
 ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";  
 ptr +="<title>LED Control</title>\n";  
 ptr +="<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}\n";  
 ptr +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;} h3 {color: #444444;margin-bottom: 50px;}\n";  
 ptr +=".button {display: block;width: 80px;background-color: #3498db;border: none;color: white;padding: 13px 30px;text-decoration: none;font-size: 25px;margin: 0px auto 35px;cursor: pointer;border-radius: 4px;}\n";  
 ptr +=".button-on {background-color: #3498db;}\n";  
 ptr +=".button-on:active {background-color: #2980b9;}\n";  
 ptr +=".button-off {background-color: #34495e;}\n";  
 ptr +=".button-off:active {background-color: #2c3e50;}\n";  
 ptr +="p {font-size: 14px;color: #888;margin-bottom: 10px;}\n";  
 ptr +="</style>\n";  
 ptr +="</head>\n";  
 ptr +="<body>\n";  
 ptr +="<h1>ESP32 Web Server</h1>\n";  
 ptr +="<h3>Using Access Point(AP) Mode</h3>\n";  
 if(led1stat)  
 {ptr +="<p>LED1 Status: ON</p><a class=\"button button-off\" href=\"/led1off\">OFF</a>\n";}  
 else  
 {ptr +="<p>LED1 Status: OFF</p><a class=\"button button-on\" href=\"/led1on\">ON</a>\n";}  
  
 if(led2stat)  
 {ptr +="<p>LED2 Status: ON</p><a class=\"button button-off\" href=\"/led2off\">OFF</a>\n";}  
 else  
 {ptr +="<p>LED2 Status: OFF</p><a class=\"button button-on\" href=\"/led2on\">ON</a>\n";}  
  
 ptr +="</body>\n";  
 ptr +="</html>\n";  
 return ptr;  
}

1. Pull Data from Website:
   1. Need an HTTPClient object to interact with a webpage.
   2. Declare an object of class **HTTPClient**. This class will make available methods that will allow us work with the HTTP functionalities without needing to encode functions to treat the low level implementation details. You can check the [header file](https://github.com/espressif/arduino-esp32/blob/master/libraries/HTTPClient/src/HTTPClient.h#L122) for a list of those methods: <https://github.com/espressif/arduino-esp32/blob/master/libraries/HTTPClient/src/HTTPClient.h#L122>
      1. HTTPClient http;
   3. Then, call the [begin](https://github.com/espressif/arduino-esp32/blob/master/libraries/HTTPClient/src/HTTPClient.h#L128) method on the previously declared object, passing as a string parameter the URL where we want to perform the HTTP request. Send the request to a testing website.
      1. Note: You can directly access the URL in a web browser to check the expected content to be returned.
      2. http.begin("<http://jsonplaceholder.typicode.com/comments?id=10>"); **//**Specify the URL
   4. To send the request, call the [GET](https://github.com/espressif/arduino-esp32/blob/master/libraries/HTTPClient/src/HTTPClient.h#L146) method on the http object.
      1. This methods receives no arguments and returns the HTTP code (success value) of the request, which we will store on a variable for error handling.
         1. Note that codes lesser than 0 are error codes from the library (you can check the list [here](https://github.com/espressif/arduino-esp32/blob/master/libraries/HTTPClient/src/HTTPClient.h#L36)) and values greater than zero are standard [HTTP return codes:](https://github.com/espressif/arduino-esp32/blob/master/libraries/HTTPClient/src/HTTPClient.h#L54) <https://github.com/espressif/arduino-esp32/blob/master/libraries/HTTPClient/src/HTTPClient.h#L36>
      2. If the code is greater than zero, print both the HTTP code returned and the response to our request.
      3. Call the [getString](https://github.com/espressif/arduino-esp32/blob/master/libraries/HTTPClient/src/HTTPClient.h#L171) method, which receives no arguments and returns a string with the response. Otherwise, we will print an error message.

**if** (httpCode > 0)

{ //Check for the returning code

String payload **=** http.getString();

Serial.println(httpCode);

Serial.println(payload);

}

**else**

{

Serial.println("Error on HTTP request");

}

* 1. Finally, we will need to call the [end](https://github.com/espressif/arduino-esp32/blob/master/libraries/HTTPClient/src/HTTPClient.cpp#L213) method, to ensure the resources are freed. It’s important that we don’t forget this call.
     1. http.end(); //Free the resources
  2. With an HTTP object, you can make the following requests at a website:
     1. POST, GET, PUT, PATCH, DELETE

1. Use OpenWeather to pull JSON data
   1. Open Weather with API syntax documentation <https://openweathermap.org/current>
      1. Need an API key to be able to use OpenWeatherMap’s API. The API key is one of the parameters we need to pass in the query string of our HTTP requests
      2. Log in to OpenWeather.
      3. Create an account and verity email.
      4. Navigate to “API Keys”
   2. Format API call:
      1. In order to ask the current weather for a specific city, we need to pass a query parameter called q. This parameter receives the city name and country code divided by comma, using ISO 3166 country codes
         1. q=Louisville,pt
      2. To finalize, we need to add the API key to our URL. This will correspond to an additional query parameter called APPID, which will have the key we have retrieved after registering on the OpenWeatherMap website. Note that query parameters are separated by an ampersand (“&“).
         1. <http://api.openweathermap.org/data/2.5/weather?q=Louisville,US,pt&APPID={yourAPIkey>}
         2. Download
      3. Example API call: <https://api.openweathermap.org/data/2.5/weather?q=Louisville,US,pt&APPID=ded70c154c37972f77d0228456d87d16>
         1. JSON format returned by OpenWeather explained:
            1. <https://openweathermap.org/current>
      4. Download and read code: <https://github.com/antmhernandez/ESP32-Basics/blob/main/openWeatherAPICall.inoo>
   3. Use the JSON returned to output weather.
      1. Parse and format JSON. (See: <https://openweathermap.org/current> )
2. Access other web pages. Pull data: <https://techtutorialsx.com/2018/03/17/esp32-arduino-getting-weather-data-from-api/>
3. HTML: See middle of report: <https://electropeak.com/learn/create-a-web-server-w-esp32/>

Day 3 – Website Data, DHT22, Google Sheets Database

Access other web pages. Pull data: <https://techtutorialsx.com/2018/03/17/esp32-arduino-getting-weather-data-from-api/>

1. How to Pull Data from a Website:
   1. Need an HTTPClient object to interact with a webpage.
   2. Declare an object of class **HTTPClient**. This class will make available methods that will allow us work with the HTTP functionalities without needing to encode functions to treat the low level implementation details. You can check the [header file](https://github.com/espressif/arduino-esp32/blob/master/libraries/HTTPClient/src/HTTPClient.h#L122) for a list of those methods: <https://github.com/espressif/arduino-esp32/blob/master/libraries/HTTPClient/src/HTTPClient.h#L122>
      1. HTTPClient http;
   3. Then, call the [begin](https://github.com/espressif/arduino-esp32/blob/master/libraries/HTTPClient/src/HTTPClient.h#L128) method on the previously declared object, passing as a string parameter the URL where we want to perform the HTTP request. Send the request to a testing website.
      1. Note: You can directly access the URL in a web browser to check the expected content to be returned.
      2. http.begin("<http://jsonplaceholder.typicode.com/comments?id=10>"); **//**Specify the URL
   4. To send the request, call the [GET](https://github.com/espressif/arduino-esp32/blob/master/libraries/HTTPClient/src/HTTPClient.h#L146) method on the http object.
      1. This methods receives no arguments and returns the HTTP code (success value) of the request, which we will store on a variable for error handling.
         1. Note that codes lesser than 0 are error codes from the library (you can check the list [here](https://github.com/espressif/arduino-esp32/blob/master/libraries/HTTPClient/src/HTTPClient.h#L36)) and values greater than zero are standard [HTTP return codes:](https://github.com/espressif/arduino-esp32/blob/master/libraries/HTTPClient/src/HTTPClient.h#L54) <https://github.com/espressif/arduino-esp32/blob/master/libraries/HTTPClient/src/HTTPClient.h#L36>
      2. If the code is greater than zero, print both the HTTP code returned and the response to our request.
      3. Call the [getString](https://github.com/espressif/arduino-esp32/blob/master/libraries/HTTPClient/src/HTTPClient.h#L171) method, which receives no arguments and returns a string with the response. Otherwise, we will print an error message.

**if** (httpCode > 0)

{ //Check for the returning code

String payload **=** http.getString();

Serial.println(httpCode);

Serial.println(payload);

}

**else**

{

Serial.println("Error on HTTP request");

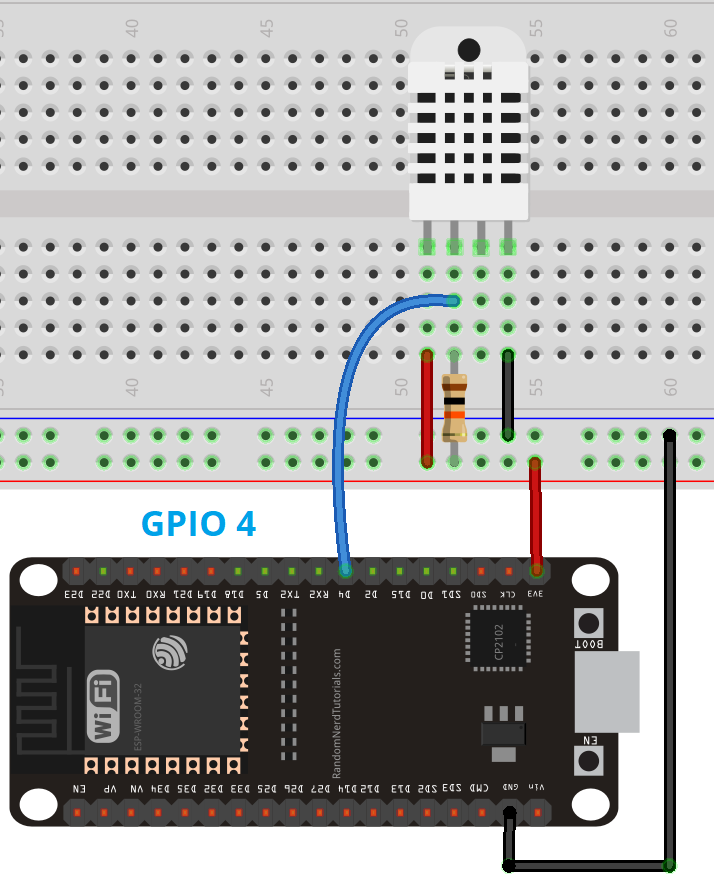
}

* 1. Finally, we will need to call the [end](https://github.com/espressif/arduino-esp32/blob/master/libraries/HTTPClient/src/HTTPClient.cpp#L213) method, to ensure the resources are freed. It’s important that we don’t forget this call.
     1. http.end(); //Free the resources
  2. With an HTTP object, you can make the following requests at a website:
     1. POST, GET, PUT, PATCH, DELETE

1. Use OpenWeather to pull JSON data
   1. Open Weather with API syntax documentation <https://openweathermap.org/current>
      1. Need an API key to be able to use OpenWeatherMap’s API. The API key is one of the parameters we need to pass in the query string of our HTTP requests
      2. Log in to OpenWeather.
      3. Create an account and verify email.
      4. Navigate to “API Keys”
   2. Format API call:
      1. In order to ask the current weather for a specific city, we need to pass a query parameter called q. This parameter receives the city name and country code divided by comma, using ISO 3166 country codes
         1. q=Louisville,pt
      2. To finalize, we need to add the API key to our URL. This will correspond to an additional query parameter called APPID, which will have the key we have retrieved after registering on the OpenWeatherMap website. Note that query parameters are separated by an ampersand (“&“).
         1. <http://api.openweathermap.org/data/2.5/weather?q=Louisville,US,pt&APPID={yourAPIkey>}
         2. Download
      3. Example API call: <https://api.openweathermap.org/data/2.5/weather?q=Louisville,US,pt&APPID=ded70c154c37972f77d0228456d87d16>
         1. JSON format returned by OpenWeather explained:
            1. <https://openweathermap.org/current>
      4. Download and read code: <https://github.com/antmhernandez/ESP32-Basics/blob/main/openWeatherAPICall.inoo>
   3. Use the JSON returned to output weather.
      1. Parse and format JSON. (See: <https://openweathermap.org/current> )
2. Wire up the DHT22. Draw readings and output to serial terminal.
   1. <https://randomnerdtutorials.com/esp32-dht11-dht22-temperature-humidity-sensor-arduino-ide/>

|  |  |
| --- | --- |
| **DHT pin** | **Connect to** |
| **1** | 3.3V |
| **2** | Any digital GPIO; also connect a 10k Ohm pull-up resistor |
| **3** | Don’t connect |
| **4** | GND |

* 1. Add a 10k pullup resistor to the board in parallel with the digital output (GPIO4) and running to 3v3.
     1. Explain Pull-Up Resistors here: <https://www.electronics-tutorials.ws/logic/pull-up-resistor.html> (first 3 sections: intro, Pull-Up Resistors, Pull-Up Resistor Application)



* 1. Add 2 Libraries: Sketch > Include Library > Manage Libraries
     1. DHT sensor library by Adafruit
     2. “Adafruit Unified Sensor” library
  2. Read Code: <https://github.com/antmhernandez/ESP32-Basics/blob/main/DHT22-Sensor-Data.ino>
  3. Copy/Paste into Arduino IDE. 9600 baud.

1. Write to Google Sheet!!! Store data. Build table and view from web page. <https://www.techiesms.com/iot-attendance-system-without-website/>
   1. Create a Google Sheet and give it a memorable title
   2. Name the sheet used (at the bottom). 1 word, no spaces, lower case.
   3. Use the first Row to name the data for each Column to be used.
      1. These will serve as pointers to push data to the Sheet.
      2. Must be one-word with NO uppercase letters. Hyphen accepted.
         1. Example: temperature-data-fahrenheit, degfahrenheit
   4. Note the Sheet ID located in the URL
      1. Example URL: [https://docs.google.com/spreadsheets/d/**1BdQzuTeYr4Tf4zwT-LP1f45rfk63oWZTrQ\_cIDfgWfgD**/edit#gid=0](https://docs.google.com/spreadsheets/d/1BdQzuTeYr4Tf4zwT-LP1f45rfk63oWZTrQ_cIDfgWfgD/edit#gid=0)
      2. Sheet ID = **1BdQzuTeYr4Tf4zwT-LP1f45rfk63oWZTrQ\_cIDfgWfgD**
   5. Create a Google App Script that will allow us to push data from the ESP to the Sheet.
      1. Extensions -> Apps Script (formerly under Tools -> Script Editor).
      2. Copy/Paste the code below:

var sheet\_id = "YOUR SHEET ID";  
var sheet\_name = "NAME OF YOUR SHEET";  
function doGet(e){  
var ss = SpreadsheetApp.openById(sheet\_id);  
var sheet = ss.getSheetByName(sheet\_name);  
var sensor = Number(e.parameter.sensor);  
var date = Number(e.parameter.date);  
sheet.appendRow([sensor,date]);  
}

* + - 1. Replace “YOUR SHEET ID” and “NAME OF YOUR SHEET”.
         1. NOTE: “Name of your sheet” is the SHEET name, not the document name.
    1. “Save.” (with name)
    2. “Deploy”.
       1. “New Deployment”: Select Type: Web app.
       2. Access as “ME” REQUIRED FOR ACCESS PERMISSION!
    3. Who can access: “Anyone”
    4. “Deploy”
    5. “Authorize Access”
       1. Select your Google account from prompt and click “Allow”.
       2. This will deploy the web app and will give you the Deployment ID and web app URL.
          1. **Please copy these and save the Deployment ID and web App URL somewhere safe.**
          2. Deployment ID:

AKfycbygFOU2t2zmLH0OLfEHPqi\_IL-qGFQGJBZ8t-PWOCPMZylGVCbv2iW61Hr8HzGNtNoM

* + - * 1. Web app URL: <https://script.google.com/macros/s/AKfycbygFOU2t2zmLH0OLfEHPqi_IL-qGFQGJBZ8t-PWOCPMZylGVCbv2iW61Hr8HzGNtNoM/exec>
        2. Apps Script URL: <https://script.google.com/macros/library/d/1qOMLhRjy7bwnt5wQdt3uoJy_ANJBFlmTJ5FrbrKPWjLvK61HqotNyqBl/3>
      1. In the case that a “**This app isn’t verified”** error is returned while authorizing, click on **advanced** and ‘Go to your ‘**Script\_name**’(unsafe).
  1. Test the Google App Script:
     1. Title the first two column headers (first row) as “date” and “sensor”.
     2. simply copy and paste the web app URL to any browser and add **?sensor=35&date=1103** to the URL after **exec**.
     3. Enter the URL in the URL bar and press enter:
        1. https://script.google.com/macros/s/AKfycbxGvdXDMgo35mujMf9kUHXX5Dh\_IVnqDAVNEl6KQjj8hF4CH7jOzB8rwKV07uT21iob/exec?humidity=5000
     4. This should return a page that says, “The script completed but did not return anything.”
     5. Return to the Google Sheet and view that the values have indeed been added.
  2. Copy/Paste the “Google-Sheets-send-data.ino” file from Github.

1. Use the DHT22 to collect temperature and humidity data. Write this to a Google Sheet.

Day 4 – Bluetooth and OTA Updates.

1. **OTA**: The ESP32 board can be encoded “Over the Air” through its Wifi connection.
   1. The additional OTA Code must be included in EACH sketch that is written to the board.
   2. The instance OTA Code is not included, the board can no longer receive OTA updates.
2. **Basic OTA** is delivered via the Arduino IDE, over Wifi, to the board.
3. **Web Updater OTA** uses functionality through a web page, served by the ESP32, to add code to the board via Wifi.

We will use the Web Updater OTA method. The Basic OTA method requires Python 2.7+, a large file, to be installed on your machine. It then uses the Arduino IDE to recognize a ‘ESP32 board port’ for upload.

1. Web Updater OTA:
   1. Download the ESP32 board add-on
      1. File -> Preferences -> Additional Boards Manager URLs: Add <https://dl.espressif.com/dl/package_esp32_index.json> (no quotes)
      2. Tools -> Board -> Board Manager: Search ESP32 by Espressif
   2. Access the Webupdater OTA code:
      1. File -> Examples -> ArduinoOTA -> OTAWebUpdater
      2. Open the file. Change the SSID and PASSWORD values to your Network’s.
      3. *(YES! Arduino comes jam-packed with example code of various kinds!!)*
      4. Upload the code
      5. Visit the IP address shown
      6. “admin” and “admin” for username and password
   3. Download the “OTA-WebUpdate-and-Blink" sketch to a New Arduino project file.
      1. Change the SSID and Password
   4. Sketch -> Export Compiled Binary
      1. This .bin file lives in the Arduino -> [Project Name] folder
      2. You will upload the .bin file to the web interface for the ESP32
      3. DO NOT FORGET TO INCLUDE THE OTA CODE. Embed additional code within it.
2. More about OTA Updates: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/ota.html>
3. Bluetooth! Host Mode: <https://randomnerdtutorials.com/esp32-bluetooth-classic-arduino-ide/>
   1. **Bluetooth** = short-range wireless technology standard for exchanging data between fixed and mobile devices over short distances (~33 ft) and for building personal area networks (PAN).
   2. **BLE** = **Bluetooth Low Energy** =
      1. Unlike Bluetooth which is always on, BLE remains in sleep mode constantly except for when a connection is initiated. This makes it consume very low power. BLE consumes approximately 1/100th as much power as Bluetooth (depending on the use case).
      2. Additionally, BLE supports not only point-to-point communication, but also broadcast mode, and mesh network. Due to its properties, BLE is suitable for applications that need to exchange small amounts of data periodically running on a coin cell

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Conventional Bluetooth** | **Low Energy Bluetooth** |
| **Communication** | Bidirectional and Continuous | Short data transfers, unidirectional |
| **Range** | Up to 100 m | Less than 100 m |
| **Energy Consumption** | Up to 1 W | 10 to 500 mW |
| **Data rate** | 1 – 3 Mbits/s | 125 kbits/s – 2 Mbits/s |
| **Latency** | 100 ms | 6 ms |
| **Voice capability** | Yes | No |

* + 1. Applications of BLE: Fitness Tracker (send small amounts of data), Indoor Positioning (GPS positioning module),

1. Load Bluetooth code in Arduino IDE:
   1. **File -> Examples -> Examples for ESP32: BluetoothSerial: SerialToSerialBT**
2. Load a Bluetooth to Serial app onto a phone or pad.
   1. nRF Connect for Mobile from Nordic, it works on [Android (Google Play Store)](https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp) and [iOS (App Store)](https://itunes.apple.com/us/app/nrf-connect/id1054362403?mt=8).
   2. Enable your bluetooth adaptor on the device which will connect
3. Download Bluetooth Terminal for ios.
   1. Navigate to Characteristic and send an ASCII message.
4. Bluetooth may “coexist” with WIFI using the “coexistance module”.
5. Other Applications of ESP32
   1. SD Card support (store and retrieve data: text, photo, audio, etc.)
   2. Audio support (microphone and speaker)

Day 5 – Create a Weather Recording Service, Blynk App??:

1. Use Google Sheets to generate a graph of the data. Create a URL link to the graph only. Pull current weather data from openweathermap.org and parse the JSON. Create a webpage, served by the ESP32 that outputs the graph URL, weather data, and current temp and humidity data.