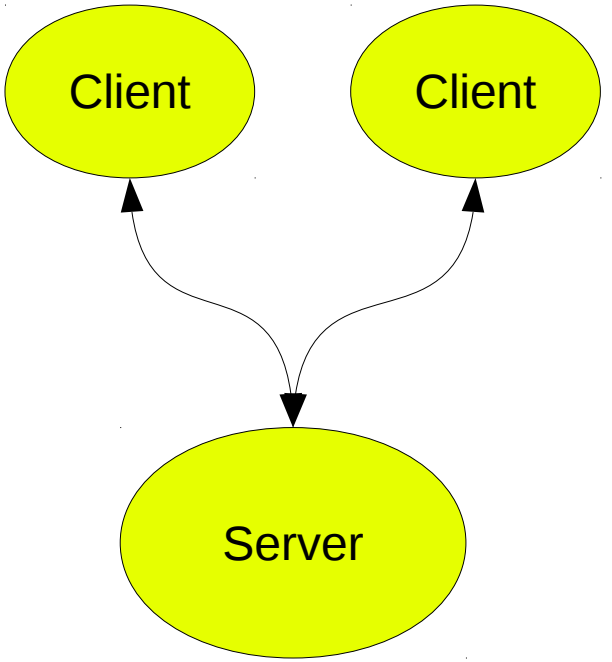
Simple Web Server

References

- www.arduino.cc
- https://github.com/esp8266/Arduino

System Design

- A web server uses the client/server software model. The server side listens for clients to connect to send information and receive information from the server.
- The client always initiates contact. Once contact is made, data can move in both directions.
- The client can be a web browser or another SLATE operating as a client.
- The client connects and sends a command. The server accepts the command, does what the command requests and sends a response.



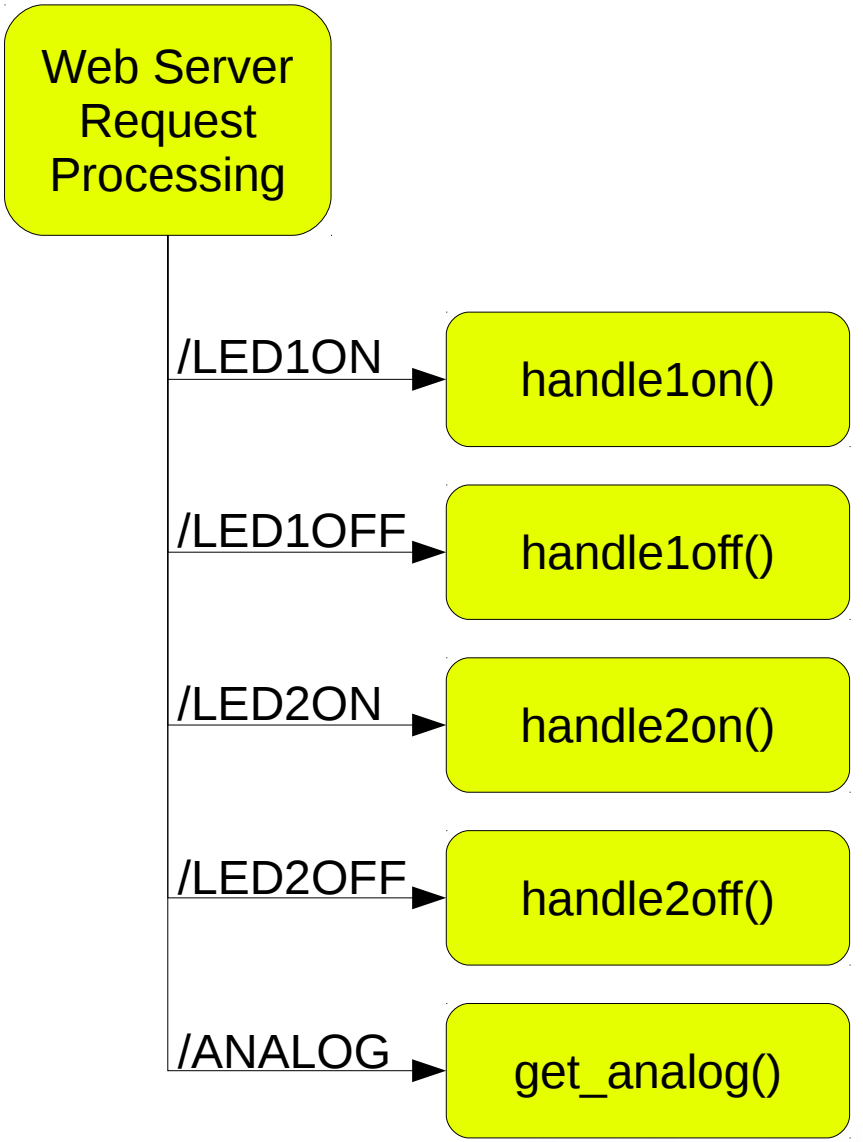
Protocol

- The client and server will use HTTP which is **H**yper **T**ext **T**ransfer **P**rotocol.
- HTTP is stateless. The client has to connect for each data transfer and then disconnect.
- The client sends a request to the server and the server responds.
- There are two request methods, GET and POST. GET will be used here.
- GET requests data from a specified resource. It is the simpler method of communicating with the server.
- Web browsers require a URL to identify a website. It is the website name or address.
- A GET request can be typed into the web browser URL location. Example:
  - 192.168.4.1/LED1ON
  - The /LED1ON identifies a resource in the web server such as a directory or file.
  - This web server will use it to execute a function.

Operation

- The web server will listen for a client to connect and send a request. If the server receives:
  - 192.168.4.1/LED1ON

  The server will call function handle1on().
- Other commands shown to the right will call the functions specified.
- The connection between the requests and the functions are configured in the setup() function.



Wiring up the SLATE

- Locate the two LEDs and the 270 ohm resistors.
- Connect digital pin 14 to the red LED.
- Connect the digital pin 15 to the green LED.
- Look in the Programming and Basic Electronics lesson on how to connect the LEDs if you cannot remember.
- The webserver code will control the LEDs.

SLATE Server Code

- The include file is a bit different. This one provides functions to support http packet processing.
- A web server object is created in the second line.
- Next are the functions that execute the commands received. Each operation is a separate function. The first four are turning LEDs on and off. The last one gets the ADC value and sends it back to the client.
- Each function sends a response indicating its operation.

```
#include <ESP8266WebServer.h>

ESP8266WebServer server(80);

void handle1on() {
  digitalWrite(14,HIGH);
  server.send(200,"text/plain","LED 1 ON\n");
}

void handle1off() {
  digitalWrite(14,LOW);
  server.send(200,"text/plain","LED 1 OFF\n");
}

void handle2on() {
  digitalWrite(15,HIGH);
  server.send(200,"text/plain","LED 1 OFF\n");
}

void handle2off() {
  digitalWrite(15,LOW);
  server.send(200,"text/plain","LED 2 OFF\n");
}

void get_analog() {
  int a = analogRead(0);
  String b = String("Result: ") + String(a) + "\n";
  server.send(200,"text/plain",b);
}
```

SLATE Code

- The setup function sets up the web server and how all the functions will be called.
- First, the serial interface is configured.
- Next, the WiFi is set to be an access point by the **mode()** function.
- The SSID is set with the **softAP()** function.
- The next 5 lines set up the webserver. Each request is assigned a function.
- The server is then started and the digital pins are configured.
- In the loop function, the function server.handleClient() is repeatedly called to handle clients that connect.

```
void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_AP);
  WiFi.softAP("weblights");
  server.on("/LED1ON",handle1on);
  server.on("/LED1OFF",handle1off);
  server.on("/LED2ON",handle2on);
  server.on("/LED2OFF",handle2off);
  server.on("/ANALOG",getanalog);
  server.begin();
  pinMode(15,OUTPUT);
  pinMode(14,OUTPUT);
}

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

Controlling the LEDs

- A web browser is required to test out the web server.
- Connect the laptop WiFi to the SLATE server using what ever SSID that was entered in the server code.
- Open a web browser.
- In the URL, enter **`192.168.4.1/LED1ON`**
- This should turn the LED on.
- Now enter **`192.168.4.1/LED1OFF`**
- The LED should turn off.
- Try the other commands.
- There should be a response in the web browser for each command.
- You can always expand the response and add html code.

Show Sensor Data

- In this section, the DHT22 temperature and humidity sensor data will be accessed through the webserver.
- First part is to have the required include files at the top of the code then declare the **server()** object and the sensor object **dht**.
- Variables for temperature and humidity are declared.
- Two long variables, **tt1** and **tt2**, are declared. Since the sensor can only be sampled every two seconds, elapsed time needs to be tracked in the **loop()** function.

```
#include <ESP8266WebServer.h>
#include <DHTesp.h>

ESP8266WebServer server(80);
DHTesp dht;

float humidity;
float temp;
unsigned long tt1,tt2;

void get_sensor()
{
  String b = "Temperature: " + String(temp) + "
Humidity: " + String(humidity); + "\n";
  server.send(200,"text/html",b);
}

void setup() {
  Serial.begin(115200);
  dht.setup(12);
  WiFi.mode(WIFI_AP);
  WiFi.softAP("sensor1");
  server.on("/",get_sensor);
  server.begin();
  tt1 = millis();
  tt2 = tt1 + 2000;
}
```

Show Sensor Data

- The get_sensor() function is called when the web browser tries to access the webserver. This function sends the temperature and humidity to the web browser as simple text.
- **server.send(200,”text/html”,b)** formats the information for the web browser to receive and sends it.
- 200 is a code indicating a valid response.

```
#include <ESP8266WebServer.h>
#include <DHTesp.h>

ESP8266WebServer server(80);
DHTesp dht;

float humidity;
float temp;
unsigned long tt1,tt2;

void get_sensor()
{
  String b = "Temperature: " + String(temp) + "
Humidity: " + String(humidity); + "\n";
  server.send(200,"text/html",b);
}

void setup() {
  Serial.begin(115200);
  dht.setup(12);
  WiFi.mode(WIFI_AP);
  WiFi.softAP("sensor1");
  server.on("/",get_sensor);
  server.begin();
  tt1 = millis();
  tt2 = tt1 + 2000;
}
```

Show Sensor Data

- The **setup()** function initializes everything. The serial interface is configured. The sensor is configured and the WiFi interface is configured.
- **server.on()** sets up the server to call the **get_sensor()** function when a web browser connects to the server.
- **tt1 = millis()** gets the time in milliseconds. Time starts at zero when the processor boots or is powered on.
- **tt2** is set to 2 seconds in the future.

```
#include <ESP8266WebServer.h>
#include <DHTesp.h>

ESP8266WebServer server(80);
DHTesp dht;

float humidity;
float temp;
unsigned long tt1,tt2;

void get_sensor()
{
  String b = "Temperature: " + String(temp) + "
Humidity: " + String(humidity); + "\n";
  server.send(200,"text/html",b);
}

void setup() {
  Serial.begin(115200);
  dht.setup(12);
  WiFi.mode(WIFI_AP);
  WiFi.softAP("sensor1");
  server.on("/",get_sensor);
  server.begin();
  tt1 = millis();
  tt2 = tt1 + 2000;
}
```

Show Sensor Data

- In the **loop()** function, the **server.handleClient()** function checks if a web browser has connected. If so, the **get_sensor()** function will get executed.
- Next, time is checked by reading time into variable **tt1**. **tt1** is then compared to **tt2** which was set 2 seconds ahead. If two seconds has elapsed, then the **if()** statement is true and the temperature and humidity are measured.
- **tt2** is then set to 2 seconds in the future again.
- This lets the **loop()** function to execute repeatedly quickly and the sensor to be accessed once every two seconds as required.

```
void loop() {
  server.handleClient();
  tt1 = millis();
  if(tt1 > tt2) {
    humidity = dht.getHumidity();
    temp = dht.getTemperature();
    Serial.print(humidity,1);
    Serial.print(" ");
    Serial.println(temp,1);
    tt2 = tt1 + 2000;
  }
}
```

Getting Sensor Data

- A web browser is required to test out the web server.
- Connect the laptop WiFi to the SLATE server using what ever SSID that was entered in the server code.
- Open a web browser.
- In the URL, enter **`192.168.4.1`**
- The temperature and humidity should be displayed in the web browser.
- You can click on the reload button to get a new result.

Auto-refreshing Webpage

- You can make the web page automatically update. It requires a little bit of html code added to the string sent to the web browser. Go to the **get_sensor()** function and add the highlighted line.
- The html code is added to the front of the measurement string and **</html>** is added to the end.
- The content=2 sets the refresh rate of the page. It can be changed to any integer value.
- Notice **\”**. You cannot have a quote inside a string which is requires quotes to identify the string. the backlash is an escape sequence that tells the compiler the next character is not to be interpreted as part of the language.

```
void get_sensor()
{
  String b = "Temperature: " + String(temp) + " Humidity: " + String(humidity); + "\n";
  b = “<html><head><meta http-equiv=\”refresh\” content=\”2\”>” + b + “</html>”;
  server.send(200,"text/html",b);
}
```

Auto Sensor Data

- Try accessing the web page again.
- Connect the laptop WiFi to the SLATE server using what ever SSID that was entered in the server code.
- Open a web browser.
- In the URL, enter **`192.168.4.1`**
- The temperature and humidity should be displayed in the web browser and should be updated every two seconds.

Other Touches

- You can change the web page background.
- The highlighted text below is added to the existing text.
- Color is specified as a combination of red, green and blue intensities.
- #ff00ff means full intensity red, no green and full intensity blue.
- The first two characters specify red, second two green and last two blue. The values are in hexidecimal. You can search the web about hexidecimal numbers.
- You can add more html code if you want. Just make sure it is inserted in front of </body>. Search the web for writing html web pages.

```
void get_sensor()
{
  String b = "Temperature: " + String(temp) + " Humidity: " + String(humidity); + "\n";
  b = “<html><head><meta http-equiv=\”refresh\” content=\”2\”><body bgcolor=\”#ff00ff\”>” + b +
“</body></html>”;
  server.send(200,"text/html",b);
}
```

Static Web Page

- In this section, we will use the file system to contain a web page and set up the web server to use the file instead of having a String variable contain the html code.
- Create a new program and name it **staticweb**.
- In the staticweb directory, create a directory called **data**.
- The directory **data** will hold the html file.

Configure Arduino IDE

- You need to configure the processor to support a file system.
- Go into the **Tools** menu and select **Flash Size**
  - If your processor is blue, select **1M (512K SPIFFS).** This will allocate 512 Kbytes for storing html and associated files.
  - If your processor is black, select 4M (3M SPIFFS). This will allocate 3 MB for storing html and associated files.

Simple HTML file

- With a text editor, create the file to the right and save it as **index.html**.
- This will create a web page with a sentence in bold.
- Select **ESP8266 Sketch Data Upload** in the Tools menu. This will create a file system image and upload it to the SLATE processor. This will take a while depending on how large of a file system was selected.

```
<html><title>Simple HTML</title><body>
<h1>This is a simple page</h1>
</body></html>
```

A nice and popular text editor is called note++ for Windows and Mac. It can be found at https://notepad-plus-plus.org/

For Mac OSX:
https://www.sublimetext.com/

Web Server Code

- Next is to create the web server. It will be mostly the same as before but not require a function to generate the web page.
- Besides starting the file system, the significant difference is the **server.serveStatic()** function. This replaces the **server.on()** function. The new function will automatically send the index.html file to the web browser when a browser connects.
- Upload the code and connect with the web browser.

```
#include <ESP8266WebServer.h>
#include <FS.h>

ESP8266WebServer server(80);

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_AP);
  WiFi.softAP("staticeweb");
  SPIFFS.begin();
  server.serveStatic("/",SPIFFS,"/index.html");
  server.begin();
}

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

Webserver Code

- Multiple web pages can be included. Each web page needs to be linked using the **server.serveStatic()** function.
- The code to the right shows the second page added to the web server. A file **page2.html** needs to be added to the **data** directory.
- Copy the **index.html** to **page2.html**. Modify the text so it is different from **index.html**.

```
#include <ESP8266WebServer.h>
#include <FS.h>

ESP8266WebServer server(80);

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_AP);
  WiFi.softAP("staticeweb");
  SPIFFS.begin();
  server.serveStatic("/",SPIFFS,"/index.html");
  server.serveStatic("/page2",SPIFFS,"page2.html");
  server.begin();
}

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

- In the browser URL, enter:
- **192.168.4.1/page2**
- The second page should come up.

Incorporating Images

- Images can be included in the web page by copying the image into the **data** directory.
- **index.html** needs to be modified to display the image in the web page.
- Find a jpeg image from the internet and download into the data directory. Make sure the image size is not too large. It has to fit in the memory.
- Add the two bold lines to **index.html**. Replace **mypic** name with the image file downloaded.
- Select the Sketch Data Upload to upload the image and html file.

```
<html><title>Simple HTML</title><body>
<h1>This is a simple page</h1>
<br>
<img src="mypic.jpg">
</body></html>
```

Updated Web Server Code

- Embedding an image is a bit more complex.
- Add the function highlighted. It opens the image file. It then sends the image to the browser so it gets shown in the web page. The image file is then closed.
- **server.on()** is added so it will call the function when the image is requested by the web browser.
- Each image will required a function and a **server.on()** function

```
#include <ESP8266WebServer.h>
#include <FS.h>

ESP8266WebServer server(80);

void send_pic() {
  File img = SPIFFS.open("/mypic.jpg","r");
  server.streamFile(img,"image/jpeg");
  img.close();
}

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_AP);
  WiFi.softAP("staticeweb");
  SPIFFS.begin();
  server.serveStatic("/",SPIFFS,"/index.html");
  server.on("/mypic.jpg",send_pic);
  server.begin();
}

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

Simple Generic Web Server

- This code will let you have as many web pages and images as can fit in the file system. The trick is to use the function **server.onNotFound()**. This function is called when no other **server.on()** functions detect a file. It is normally used to send an error message to the web browser.

```
#include <ESP8266WebServer.h>
#include <FS.h>

ESP8266WebServer server(80);

void handleOther()
{
  String path = server.uri();
  String dataType = "text/plain";
  if(path.endsWith("/")) path = "/index.html";
  if(path.endsWith(".jpg")) dataType = "image/jpeg";
  else if(path.endsWith(".png")) dataType = "image/png";
  else if(path.endsWith(".html")) dataType = "text/html";
  Serial.println(path.c_str());
  File datafile = SPIFFS.open(path.c_str(),"r");
  Serial.println(datafile);
  server.streamFile(datafile,dataType);
  datafile.close();
}

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_AP);
  WiFi.softAP("simpleweb");
  SPIFFS.begin();
  server.onNotFound(handleOther);
  server.begin();
}

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

Simple Generic Web Server

- **server.onNotFound()** calls the function **handleOther()**.
- **handleOther()** looks at the URL information received from the browser and determines what file to send.
- **server.uri()** gets the path of what is requested. The path is printed to the serial monitor. The function **path.c_str()** converts path to a printable string.

```
#include <ESP8266WebServer.h>
#include <FS.h>

ESP8266WebServer server(80);

void handleOther()
{
  String path = server.uri();
  Serial.println(path.c_str());
  String dataType = "text/plain";
  if(path.endsWith("/")) path = "/index.html";
  if(path.endsWith(".jpg")) dataType = "image/jpeg";
  else if(path.endsWith(".png")) dataType = "image/png";
  else if(path.endsWith(".html")) dataType = "text/html";
  Serial.println(path.c_str());
  File datafile = SPIFFS.open(path.c_str(),"r");
  server.streamFile(datafile,dataType);
  datafile.close();
}

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_AP);
  WiFi.softAP("simpleweb");
  SPIFFS.begin();
  server.onNotFound(handleOther);
  server.begin();
}

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

Simple Generic Web Server

- variable **dataType** is set to **“text/plain”**. This is required by the web browser to indicate the type of information being received. Each file type has a unique data type.
- The first check is to determine if the main page is to be accessed. This is usually indicated by */*. If it is, then the path variable is set to **/index.html**. This will cause the function to send the **index.html** page to the web browser.

```
#include <ESP8266WebServer.h>
#include <FS.h>

ESP8266WebServer server(80);

void handleOther()
{
  String path = server.uri();
  Serial.println(path.c_str());
  String dataType = "text/plain";
  if(path.endsWith("/")) path = "/index.html";
  if(path.endsWith(".jpg")) dataType = "image/jpeg";
  else if(path.endsWith(".png")) dataType = "image/png";
  else if(path.endsWith(".html")) dataType = "text/html";
  else server.send(404,dataType,”Error”);
  Serial.println(path.c_str());
  File datafile = SPIFFS.open(path.c_str(),"r");
  server.streamFile(datafile,dataType);
  datafile.close();
}

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_AP);
  WiFi.softAP("simpleweb");
  SPIFFS.begin();
  server.onNotFound(handleOther);
  server.begin();
}

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

Simple Generic Web Server

- Next, file types are checked. **path.endsWith()** will check if the argument matches the end of the string variable value. **dataType** is set to the type of file being accessed.

```
#include <ESP8266WebServer.h>
#include <FS.h>

ESP8266WebServer server(80);

void handleOther()
{
  String path = server.uri();
  Serial.println(path.c_str());
  String dataType = "text/plain";
  if(path.endsWith("/")) path = "/index.html";
  if(path.endsWith(".jpg")) dataType = "image/jpeg";
  else if(path.endsWith(".png")) dataType = "image/png";
  else if(path.endsWith(".html")) dataType = "text/html";
  else server.send(404,dataType,"Error");
  File datafile = SPIFFS.open(path.c_str(),"r");
  if(!datafile)
    server.send(404,dataType,"file not found");
  server.streamFile(datafile,dataType);
  datafile.close();
}

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_AP);
  WiFi.softAP("simpleweb");
  SPIFFS.begin();
  server.onNotFound(handleOther);
  server.begin();
}

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

Simple Generic Web Server

- If the end of the file does not match anything, then an error message is sent.
- If there was no error, the requested data file is opened. **if(!datafile)** checks to see if the file exists. If not, an error message is sent. Notice in the second error message, the data type is spelled out. This is because dataType could have been changed to something else and the error message is plain text.

```
#include <ESP8266WebServer.h>
#include <FS.h>

ESP8266WebServer server(80);

void handleOther()
{
  String path = server.uri();
  Serial.println(path.c_str());
  String dataType = "text/plain";
  if(path.endsWith("/")) path = "/index.html";
  if(path.endsWith(".jpg")) dataType = "image/jpeg";
  else if(path.endsWith(".png")) dataType = "image/png";
  else if(path.endsWith(".html")) dataType = "text/html";
  else server.send(404,dataType,"Error");
  File datafile = SPIFFS.open(path.c_str(),"r");
  if(!datafile)
    server.send(404,"text/plain","file not found");
  server.streamFile(datafile,dataType);
  datafile.close();
}

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_AP);
  WiFi.softAP("simpleweb");
  SPIFFS.begin();
  server.onNotFound(handleOther);
  server.begin();
}

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

Simple Generic Web Server

- Finally, if there were no errors, the selected file is sent and the file is then closed.
- This program will let you create a web server with multiple web pages and multiple images embedded in the web pages.
- Just remember all the web pages and images must fit within the file system space. Any image or file update requires the Sketch Data Upload. If all the files exceed the size of the file system, the upload will let you know.

```
#include <ESP8266WebServer.h>
#include <FS.h>

ESP8266WebServer server(80);

void handleOther()
{
  String path = server.uri();
  Serial.println(path.c_str());
  String dataType = "text/plain";
  if(path.endsWith("/")) path = "/index.html";
  if(path.endsWith(".jpg")) dataType = "image/jpeg";
  else if(path.endsWith(".png")) dataType = "image/png";
  else if(path.endsWith(".html")) dataType = "text/html";
  else server.send(404,dataType,"Error");
  File datafile = SPIFFS.open(path.c_str(),"r");
  if(!datafile)
    server.send(404,"text/plain","file not found");
  server.streamFile(datafile,dataType);
  datafile.close();
}

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_AP);
  WiFi.softAP("simpleweb");
  SPIFFS.begin();
  server.onNotFound(handleOther);
  server.begin();
}

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

Learning HTML

- If you want to learn how to create web pages with html, do a search for learn html. Videos will come up.
- Other possible sites:
  - https://www.codecademy.com/learn/learn-html
  - https://www.w3schools.com/Html/