Serveur Web ESP32 - IDE Arduino

Dans ce projet, vous créez un serveur Web autonome avec un ESP32 qui contrôle les sorties (deux voyants) à l'aide de l'environnement de programmation Arduino IDE. Le serveur Web est réactif pour les mobiles et peut être accédé avec n'importe quel appareil avec un navigateur sur le réseau local. Nous allons vous montrer comment créer le serveur Web et comment le code fonctionne pas à pas.

Aperçu du projet

Avant de passer directement au projet, il est important de définir les tâches de notre serveur Web afin de pouvoir suivre plus facilement les étapes suivantes.

Le serveur Web que vous allez créer contrôle deux voyants connectés aux ESPIO GPIO 26 et GPIO 27 ;

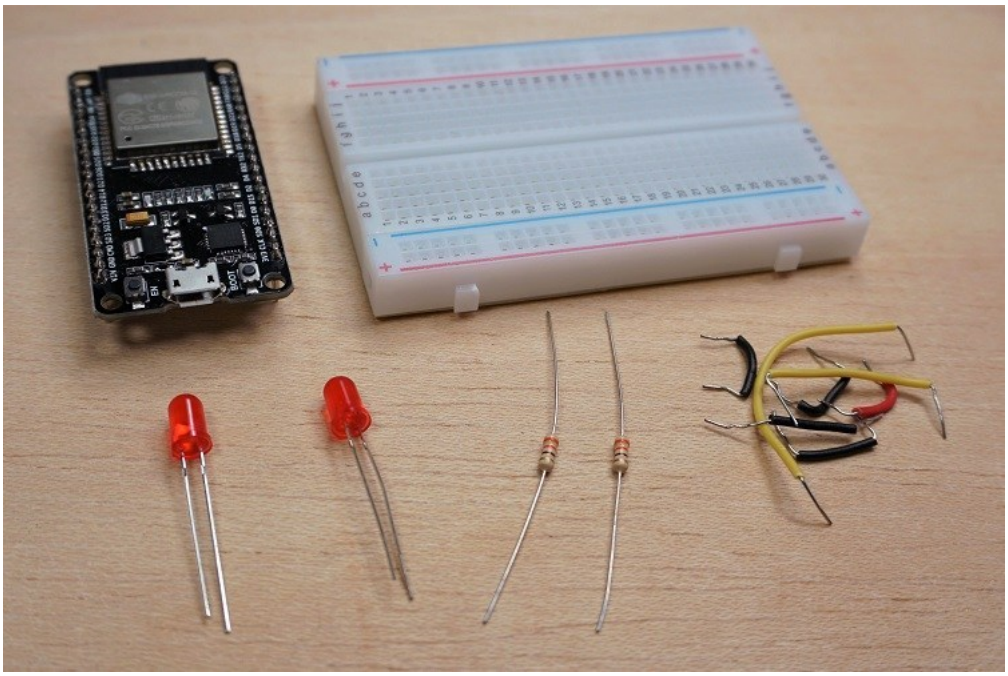
Vous pouvez accéder au serveur Web ESP32 en saisissant l'adresse IP ESP32 dans un navigateur du réseau local.

En cliquant sur les boutons de votre serveur Web, vous pouvez changer instantanément l'état de chaque voyant.

Ceci est juste un exemple simple pour illustrer comment construire un serveur Web qui contrôle les sorties. L'idée est de remplacer ces voyants par un relais ou tout autre composant électronique de votre choix pour d'autres applications.

Pièces requises

Pour ce tutoriel, vous aurez besoin des éléments suivants:

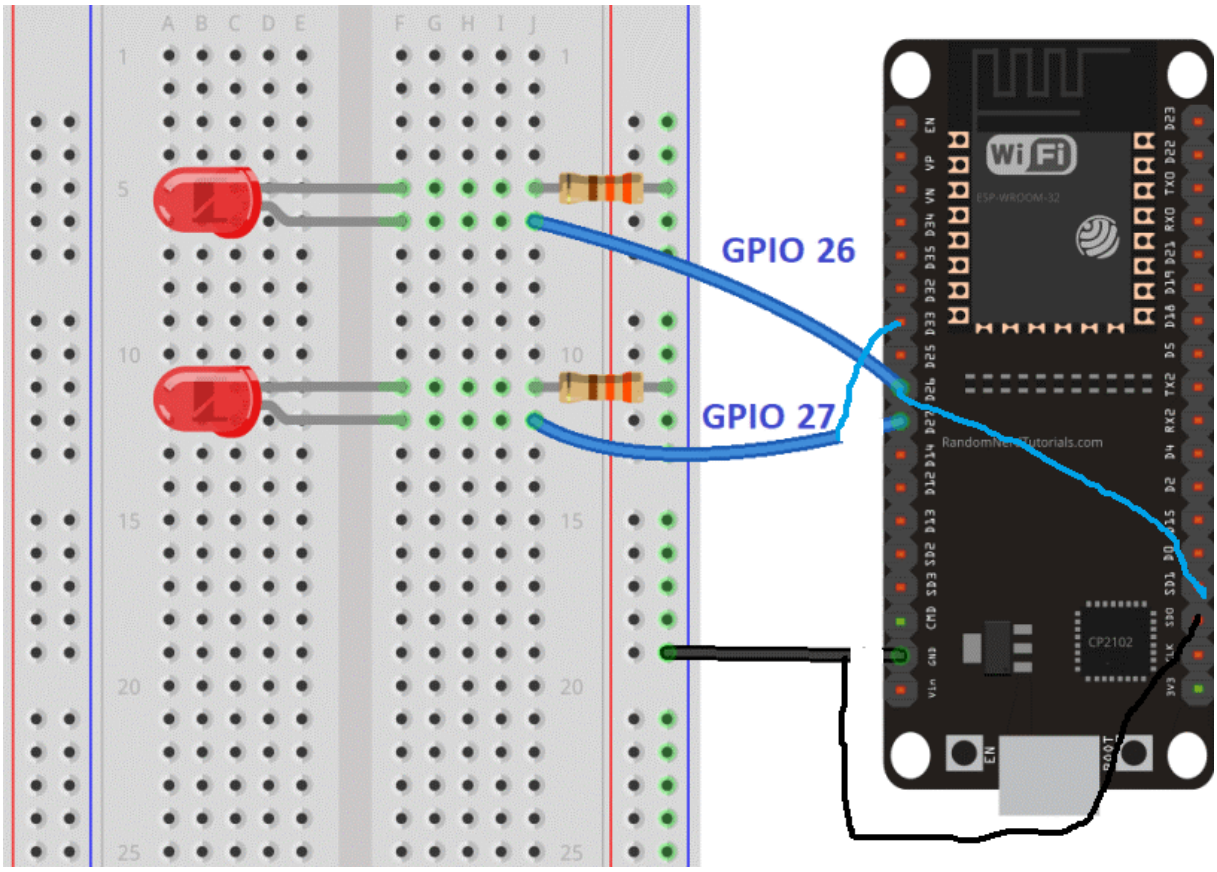


- Carte de développement ESP32
- 2x 5mm LED
- 2x 330 Ohm résistance
- Plaque à trous
- Fils de connexion

Schématique

Commencez par construire le circuit. Connectez deux voyants à l'ESP32 comme indiqué dans le schéma suivant: l'un connecté à GPIO 26 et l'autre à GPIO 27.

Remarque : nous utilisons la carte ESP32 feather. Avant d'assembler le circuit, assurez-vous de vérifier le brochage de la carte que vous utilisez.



Télécharger le code

Maintenant, vous pouvez télécharger le code et le serveur Web fonctionnera immédiatement. Suivez les étapes suivantes pour télécharger le code sur l'ESP32.

Définition de vos identifiants réseau

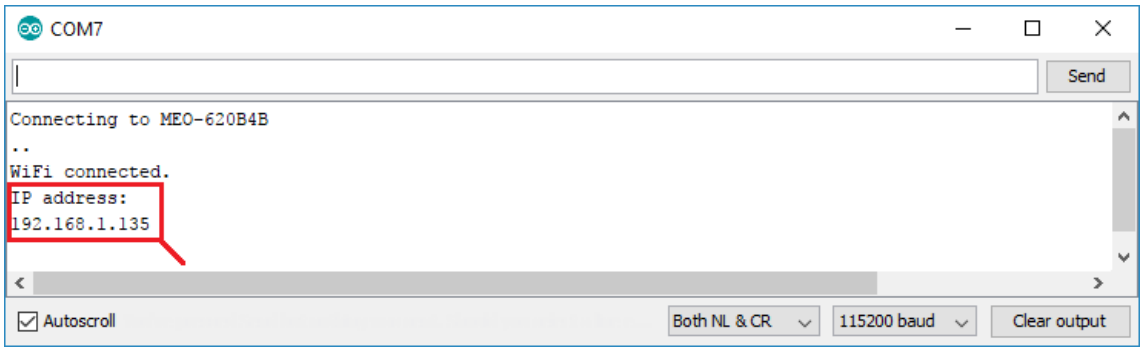
Vous devez modifier les lignes suivantes avec vos informations d'identification réseau: SSID et mot de passe. Le code est bien commenté sur l'endroit où vous devez apporter les modifications.

```
// Replace with your network credentials
const char* ssid     = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

Recherche de l'adresse IP de l'ESP

Après avoir chargé le code, ouvrez le moniteur série à une vitesse de transmission de 115200 bauds.

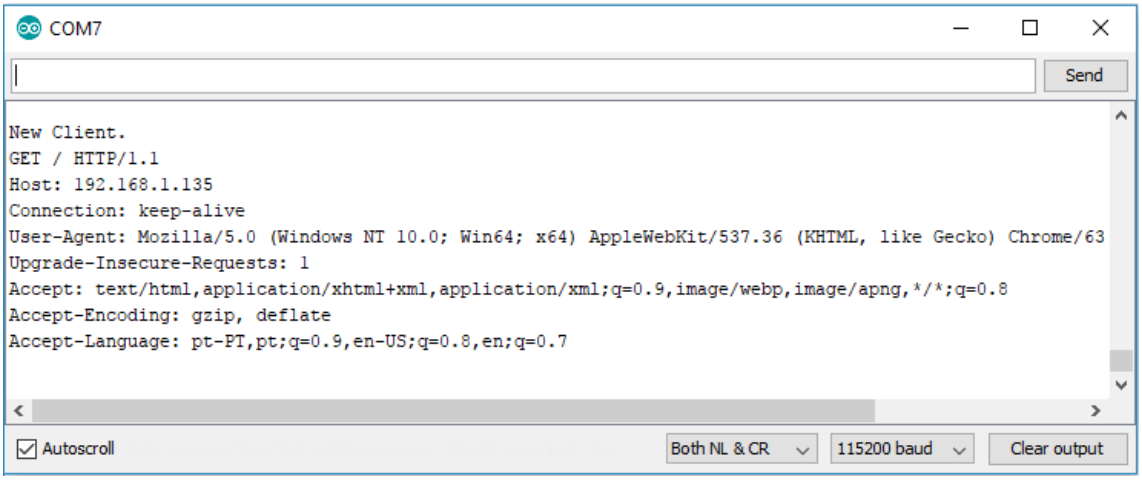
Appuyez sur la touche ESP32 EN (réinitialisation). L'ESP32 se connecte au Wi-Fi et transmet l'adresse IP ESP sur le moniteur série. Copiez cette adresse IP, car vous en avez besoin pour accéder au serveur Web ESP32.



Accéder au serveur Web

Pour accéder au serveur Web, ouvrez votre navigateur, collez l'adresse IP ESP32 et vous verrez la page suivante. Dans notre cas, il s'agit de **192.168.1.135** .

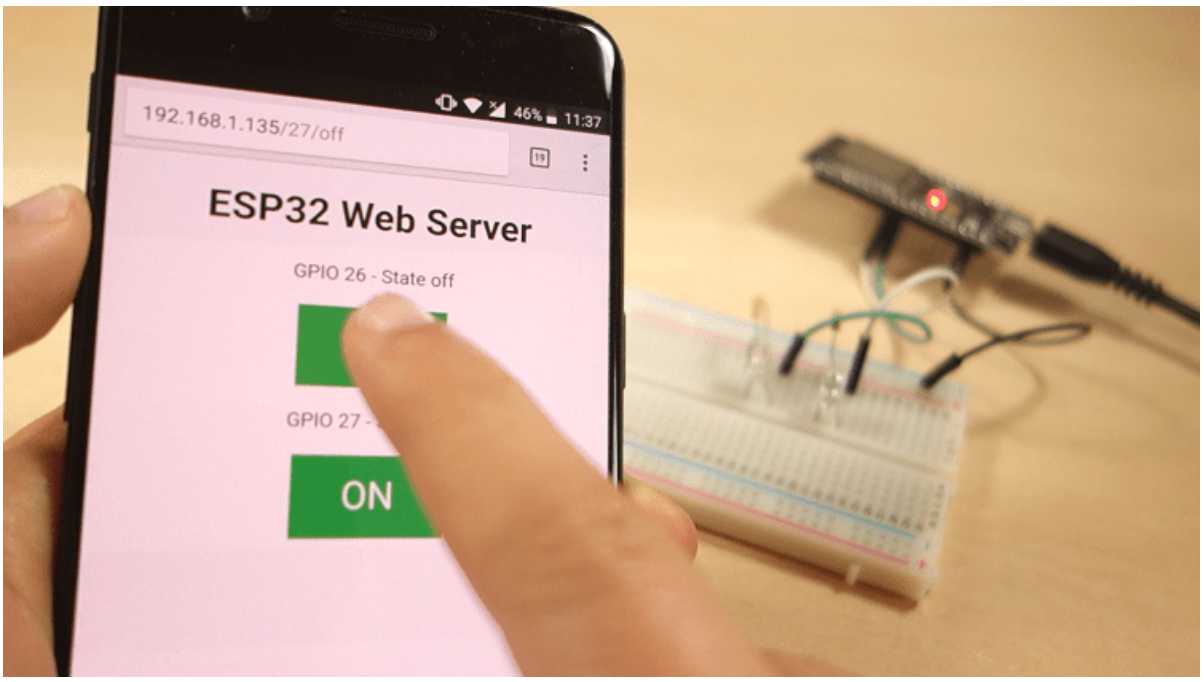
Si vous regardez le moniteur de série, vous pouvez voir ce qui se passe en arrière-plan. L'ESP reçoit une requête HTTP d'un nouveau client (dans ce cas, votre navigateur).



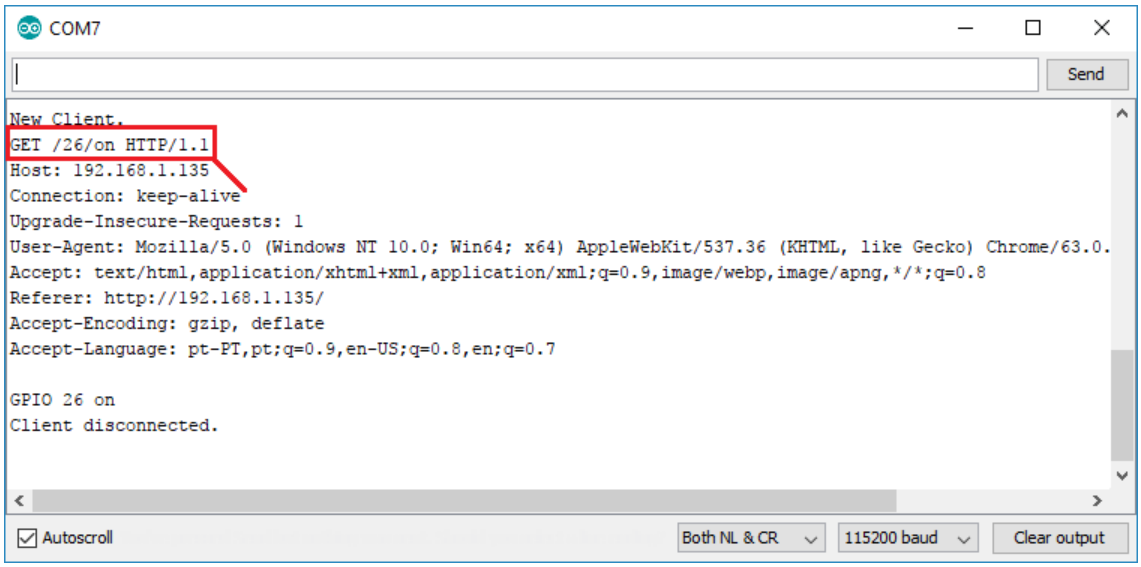
Vous pouvez également voir d'autres informations sur la requête HTTP.

Test du serveur Web

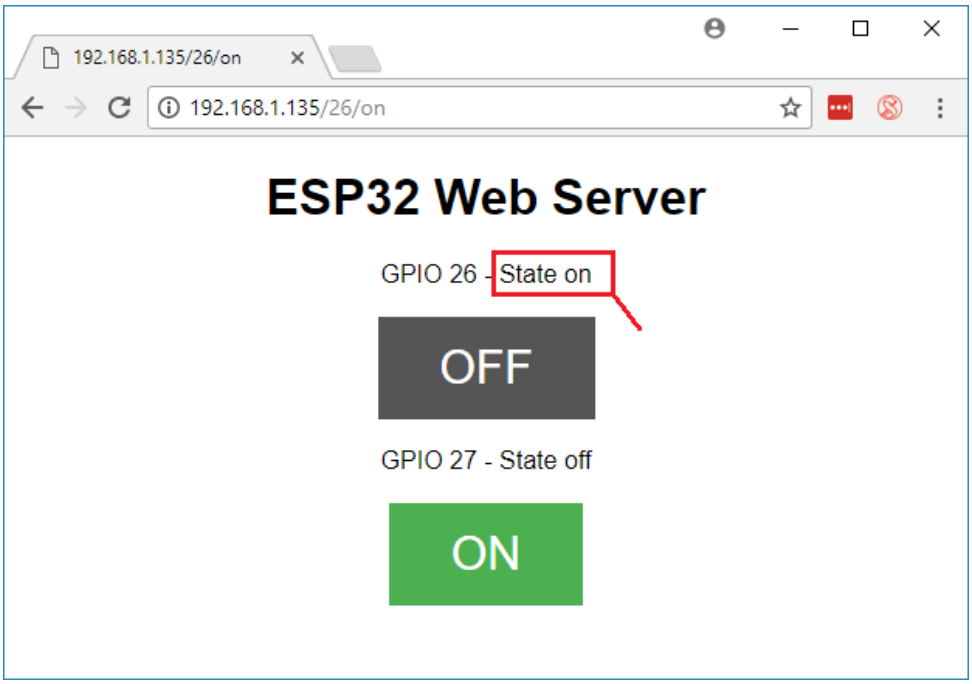
Vous pouvez maintenant tester si votre serveur Web fonctionne correctement. Cliquez sur les boutons pour contrôler les voyants.



En même temps, vous pouvez jeter un coup d'œil sur le moniteur série pour voir ce qui se passe en arrière-plan. Par exemple, lorsque vous cliquez sur le bouton pour activer GPIO 26, l'ESP32 reçoit une demande sur l' URL **/ 26 / on** .



Lorsque l'ESP32 reçoit cette demande, il allume le voyant associé à GPIO 26 et met à jour son état sur la page Web.



Le bouton pour GPIO 27 fonctionne de manière similaire. Vérifiez qu'il fonctionne correctement.

Comment fonctionne le code

Dans cette section, nous examinerons le code de plus près pour voir comment cela fonctionne.

La première chose à faire est d'inclure la bibliothèque WiFi.

```
#include <WiFi.h>
```

Comme mentionné précédemment, vous devez insérer votre identifiant SSID et votre mot de passe dans les lignes suivantes, à l'intérieur des guillemets.

```
const char* ssid = "";

const char* password = "";
```

Ensuite, vous configurez votre serveur Web sur le port 80.

```
WiFiServer server(80);
```

La ligne suivante crée une variable pour stocker l'en-tête de la requête HTTP:

```
String header;
```

Ensuite, vous créez des variables auxiliaires pour stocker l'état actuel de vos sorties. Si vous souhaitez ajouter plus de sorties et sauvegarder son état, vous devez créer plus de variables.


```
String output26State = "off";

String output27State = "off";
```

Vous devez également affecter un GPIO à chacune de vos sorties. Nous utilisons ici GPIO 26 et GPIO 27 . Vous pouvez utiliser n'importe quel autre GPIO approprié.

```
const int output26 = 26;

const int output27 = 27;
```

setup()

Maintenant, passons à la `setup()`. Premièrement, nous commençons une communication série à un débit de 115200 bauds à des fins de débogage.

```
Serial.begin(115200);
```

Vous définissez également vos GPIO en tant que OUTPUT et les définissez sur LOW.

```
// Initialize the output variables as outputs

pinMode(output26, OUTPUT);

pinMode(output27, OUTPUT);


// Set outputs to LOW

digitalWrite(output26, LOW);

digitalWrite(output27, LOW);
```

Les lignes suivantes commencent la connexion Wi-Fi par `WiFi.begin (ssid, mot de passe)` , attendent la connexion établie et impriment l'adresse IP ESP dans le moniteur de série.

```
// Connect to Wi-Fi network with SSID and password

Serial.print("Connecting to ");

Serial.println(ssid);
```

```
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}
// Print local IP address and start web server
Serial.println("");
Serial.println("WiFi connected.");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
server.begin();
```

loop()

Dans la loop(), nous programmons ce qui se produit lorsqu'un nouveau client établit une connexion avec le serveur Web.

L'ESP32 est toujours à l'écoute des clients entrants avec la ligne suivante:

```
WiFiClient client = server.available(); // Listen for incoming clients
```

Lorsqu'une demande est reçue d'un client, nous enregistrons les données entrantes. La boucle while qui suit sera exécutée tant que le client reste connecté. Nous ne recommandons pas de changer la partie suivante du code à moins que vous ne sachiez exactement ce que vous faites.

```
if (client) { // If a new client connects,
  Serial.println("New Client."); // print a message out in the serial port
  String currentLine = ""; // make a String to hold incoming data from the client
  while (client.connected()) { // loop while the client's connected
    if (client.available()) { // if there's bytes to read from the client,
      char c = client.read(); // read a byte, then
      Serial.write(c); // print it out the serial monitor
      header += c;
      if (c == '\n') { // if the byte is a newline character
      // if the current line is blank, you got two newline characters in a row.
      / that's the end of the client HTTP request, so send a response:
```

```
        if (currentLine.length() == 0) {
        // HTTP headers always start with a response code (e.g.
HTTP/1.1 200 OK)
        // and a content-type so the client knows what's coming, then a
blank line:
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();
```

La section suivante des instructions if et else vérifie quel bouton a été utilisé dans votre page Web et contrôle les sorties en conséquence. Comme nous l'avons vu précédemment, nous faisons une demande sur différentes URL en fonction du bouton enfoncé.

```
// turns the GPIOs on and off
if (header.indexOf("GET /26/on") >= 0) {
  Serial.println("GPIO 26 on");
  output26State = "on";
  digitalWrite(output26, HIGH);
} else if (header.indexOf("GET /26/off") >= 0) {
  Serial.println("GPIO 26 off");
  output26State = "off";
  digitalWrite(output26, LOW);
} else if (header.indexOf("GET /27/on") >= 0) {
  Serial.println("GPIO 27 on");
  output27State = "on";
  digitalWrite(output27, HIGH);
} else if (header.indexOf("GET /27/off") >= 0) {
  Serial.println("GPIO 27 off");
  output27State = "off";
  digitalWrite(output27, LOW);
}
```

Par exemple, si vous avez appuyé sur le bouton GPIO 26 ON, l'ESP32 reçoit une demande sur l' **URL / 26 / ON** (nous pouvons voir cette information sur l'en-tête HTTP du moniteur de série). Nous

pouvons donc vérifier si l'en-tête contient l'expression **GET / 26 / on** . S'il en contient une, la variable `output26state` est `définie` sur ON et l'ESP32 allume le voyant.

Cela fonctionne de la même manière pour les autres boutons. Donc, si vous voulez ajouter plus de sorties, vous devez modifier cette partie du code pour les inclure.

Affichage de la page Web HTML

La prochaine chose que vous devez faire est de créer la page Web. L'ESP32 enverra une réponse à votre navigateur avec du code HTML pour créer la page Web.

La page Web est envoyée au client en utilisant cette expression `client.println ()` . Vous devez entrer ce que vous voulez envoyer au client sous forme d'argument.

La première chose à envoyer est toujours la ligne suivante, qui indique que nous envoyons du HTML.

```
<!DOCTYPE HTML><html>
```

Ensuite, la ligne suivante rend la page Web réactive dans n'importe quel navigateur Web.

```
client.println("<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">");
```


Et ce qui suit est utilisé pour empêcher les requêtes sur le favicon. - Vous n'avez pas à vous soucier de cette ligne.

```
client.println("<link rel=\"icon\" href=\"data:,\">");
```

Styliser la page Web

Ensuite, nous avons du texte CSS pour styliser les boutons et l'apparence de la page Web. Nous choisissons la police Helvetica, définissons le contenu à afficher sous forme de bloc et aligné au centre.

```
client.println("<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}");
```

Nous appelons nos boutons avec la couleur # 4CAF50, sans bordure, le texte est en blanc et avec ce rembourrage: 16px 40px. Nous définissons également la décoration de texte sur none, définissons la taille de la police, la marge et le curseur sur un pointeur.

```
client.println(".button { background-color: #4CAF50; border: none; color: white; padding: 16px 40px;");
```

```
client.println("text-decoration: none; font-size: 30px; margin: 2px; cursor: pointer;}");
```

Nous définissons également le style d'un second bouton, avec toutes les propriétés du bouton que nous avons définies précédemment, mais avec une couleur différente. Ce sera le style pour le bouton off.

```
client.println(".button2 {background-color: #555555;}</style></head>");
```

Définition du premier titre de la page Web

Dans la ligne suivante, vous pouvez définir le premier en-tête de votre page Web. Ici, nous avons « **Serveur Web ESP32** », mais vous pouvez modifier ce texte comme bon vous semble.

```
// Web Page Heading

client.println("<h1>ESP32 Web Server</h1>");
```

Affichage des boutons et de l'état correspondant

Ensuite, vous écrivez un paragraphe pour afficher l' état actuel de GPIO 26 . Comme vous pouvez le constater, nous utilisons la variable `output26State` , afin que l'état se `mette` à jour instantanément lorsque cette variable change.

```
client.println("<p>GPIO 26 - State " + output26State + "</p>");
```


Ensuite, nous affichons le bouton d'activation ou de désactivation, en fonction de l'état actuel du GPIO. Si l'état actuel du GPIO est désactivé, nous affichons le bouton ON, sinon nous affichons le bouton OFF.

```
if (output26State=="off") {

  client.println("<p><a href=\"/26/on\"><button
class=\"button\">ON</button></a></p>");

} else {

  client.println("<p><a href=\"/26/off\"><button class=\"button
button2\">OFF</button></a></p>");

}
```

Nous utilisons la même procédure pour GPIO 27.

Fermer la connexion

Enfin, lorsque la réponse est terminée, nous effaçons la variable d'en- `tête` et arrêtons la connexion avec le client avec `client.stop ()` .

```
// Clear the header variable

header = "";

// Close the connection

client.stop();
```

Conclusion

Dans ce tutoriel, nous vous avons montré comment construire un serveur Web avec l'ESP32. Nous vous avons montré un exemple simple de contrôle de deux voyants, mais l’idée est de remplacer ces voyants par un relais ou toute autre sortie que vous souhaitez contrôler.

Source

https://randomnerdtutorials.com/esp32-web-server-arduino-ide/

```
/*********
  Rui Santos
  Complete project details at http://randomnerdtutorials.com
*********/

// Load Wi-Fi library
#include <WiFi.h>

// Replace with your network credentials
const char* ssid     = "";
const char* password = "";

// Set web server port number to 80
WiFiServer server(80);

// Variable to store the HTTP request
String header;

// Auxiliar variables to store the current output state
String output26State = "off";
String output27State = "off";

// Assign output variables to GPIO pins
const int output26 = 26;
const int output27 = 27;

void setup() {
  Serial.begin(115200);
  // Initialize the output variables as outputs
  pinMode(output26, OUTPUT);
  pinMode(output27, OUTPUT);
  // Set outputs to LOW
  digitalWrite(output26, LOW);
  digitalWrite(output27, LOW);

  // Connect to Wi-Fi network with SSID and password
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  // Print local IP address and start web server
  Serial.println("");
  Serial.println("WiFi connected.");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  server.begin();
}

void loop(){
  WiFiClient client = server.available();   // Listen for incoming clients

  if (client) {                             // If a new client connects,
    Serial.println("New Client.");          // print a message out in the
serial port
    String currentLine = "";                // make a String to hold
incoming data from the client
    while (client.connected()) {            // loop while the client's
connected
```

```
      if (client.available()) {             // if there's bytes to read
from the client,
        char c = client.read();             // read a byte, then
        Serial.write(c);                    // print it out the serial
monitor
        header += c;
        if (c == '\n') {                    // if the byte is a newline
character
          // if the current line is blank, you got two newline characters
in a row.
          // that's the end of the client HTTP request, so send a response:
          if (currentLine.length() == 0) {
            // HTTP headers always start with a response code (e.g.
HTTP/1.1 200 OK)
            // and a content-type so the client knows what's coming, then a
blank line:
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();

            // turns the GPIOs on and off
            if (header.indexOf("GET /26/on") >= 0) {
              Serial.println("GPIO 26 on");
              output26State = "on";
              digitalWrite(output26, HIGH);
            } else if (header.indexOf("GET /26/off") >= 0) {
              Serial.println("GPIO 26 off");
              output26State = "off";
              digitalWrite(output26, LOW);
            } else if (header.indexOf("GET /27/on") >= 0) {
              Serial.println("GPIO 27 on");
              output27State = "on";
              digitalWrite(output27, HIGH);
            } else if (header.indexOf("GET /27/off") >= 0) {
              Serial.println("GPIO 27 off");
              output27State = "off";
              digitalWrite(output27, LOW);
            }

            // Display the HTML web page
            client.println("<!DOCTYPE html><html>");
            client.println("<head><meta name=\"viewport\"
content=\"width=device-width, initial-scale=1\">");
            client.println("<link rel=\"icon\" href=\"data:,\">");
            // CSS to style the on/off buttons
            // Feel free to change the background-color and font-size
attributes to fit your preferences
            client.println("<style>html { font-family: Helvetica; display:
inline-block; margin: 0px auto; text-align: center;}");
            client.println(".button { background-color: #4CAF50; border:
none; color: white; padding: 16px 40px;");
            client.println("text-decoration: none; font-size: 30px; margin:
2px; cursor: pointer;}");
            client.println(".button2 {background-color:
#555555;}</style></head>");

            // Web Page Heading
            client.println("<body><h1>ESP32 Web Server</h1>");

            // Display current state, and ON/OFF buttons for GPIO 26
```

```
            client.println("<p>GPIO 26 - State " + output26State + "</p>");
            // If the output26State is off, it displays the ON button
            if (output26State=="off") {
              client.println("<p><a href=\"/26/on\"><button
class=\"button\">ON</button></a></p>");
            } else {
              client.println("<p><a href=\"/26/off\"><button class=\"button
button2\">OFF</button></a></p>");
            }

            // Display current state, and ON/OFF buttons for GPIO 27
            client.println("<p>GPIO 27 - State " + output27State + "</p>");
            // If the output27State is off, it displays the ON button
            if (output27State=="off") {
              client.println("<p><a href=\"/27/on\"><button
class=\"button\">ON</button></a></p>");
            } else {
              client.println("<p><a href=\"/27/off\"><button class=\"button
button2\">OFF</button></a></p>");
            }
            client.println("</body></html>");

            // The HTTP response ends with another blank line
            client.println();
            // Break out of the while loop
            break;
          } else { // if you got a newline, then clear currentLine
            currentLine = "";
          }
        } else if (c != '\r') {  // if you got anything else but a carriage
return character,
          currentLine += c;      // add it to the end of the currentLine
        }
      }
    }
    // Clear the header variable
    header = "";
    // Close the connection
    client.stop();
    Serial.println("Client disconnected.");
    Serial.println("");
  }
}
```