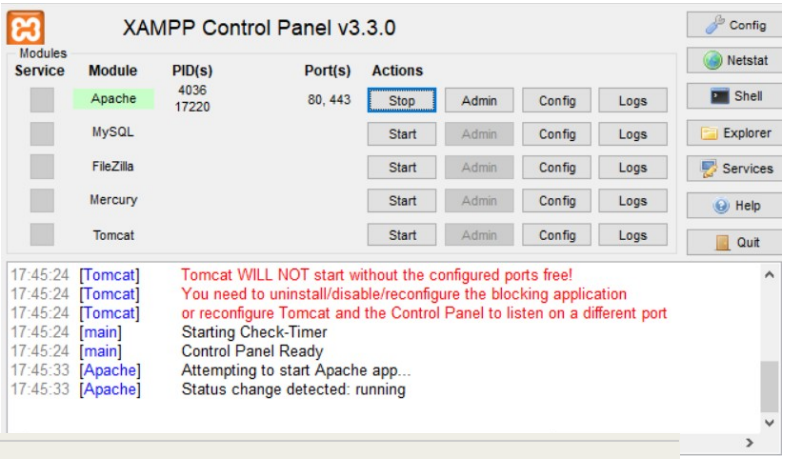
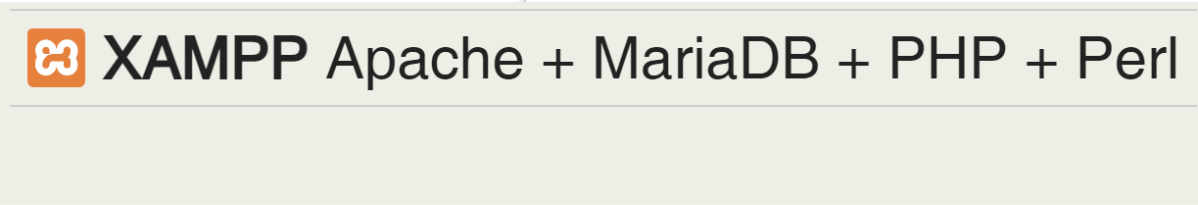
WEBSERVER CON ESP8266

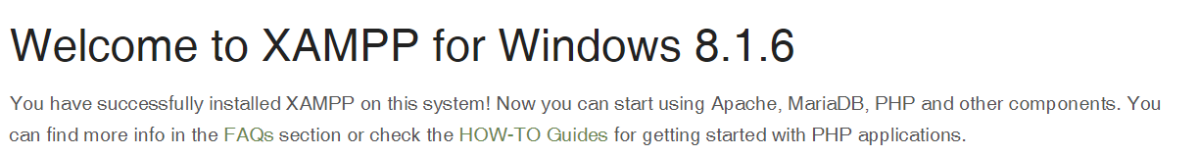
Per realizzare un Webserver con un qualsiasi PC o SBC occorre installare un apposito pacchetto software spesso abbinato con altri software come PHPmyadmin e MYSQL. E' questo il caso di XAMPP per Windows o LAMP per Linux. In entrambi i casi oltre al database MYSQL ed al software necessario per amministrarlo come PHPmyadmin, viene installato il Webserver APACHE.

Installare un Webserver su un PC significa dotarlo della struttura necessaria per comunicare con un client mediante il protocollo HTTP o HTTPS, pertanto il PC avrà un software sempre attivo chiamato "demone" che sarà in ascolto sulla porta 80 per l'HTTP o 443 per l'HTTPS e che risponderà alle richieste del client, fornendogli ad esempio il codice di una pagina WEB.

Installando ad esempio XAMPP sul proprio PC ed attivando il Webserver APACHE tramite il pannello di controllo di XAMPP, avremo che, digitando l'indirizzo del nostro computer da un altro PC connesso alla stessa rete, il nostro computer risponderà alla richiesta con la pagina di benvenuto del software XAMPP.



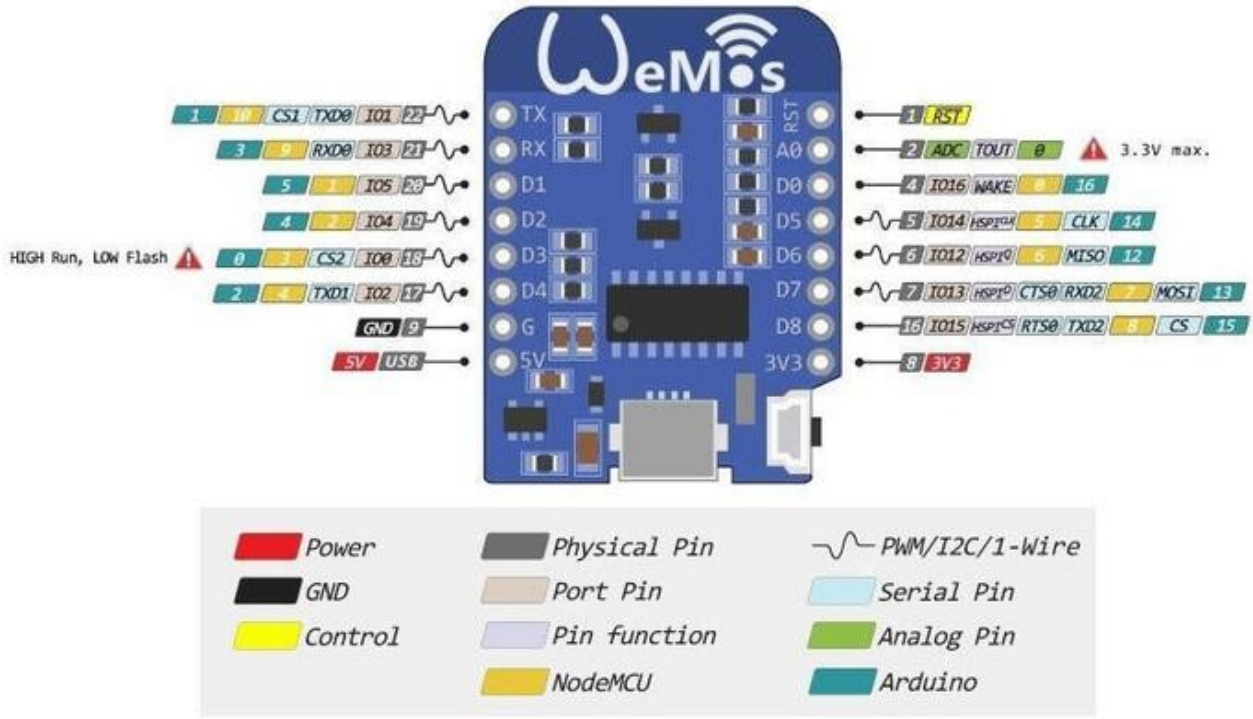




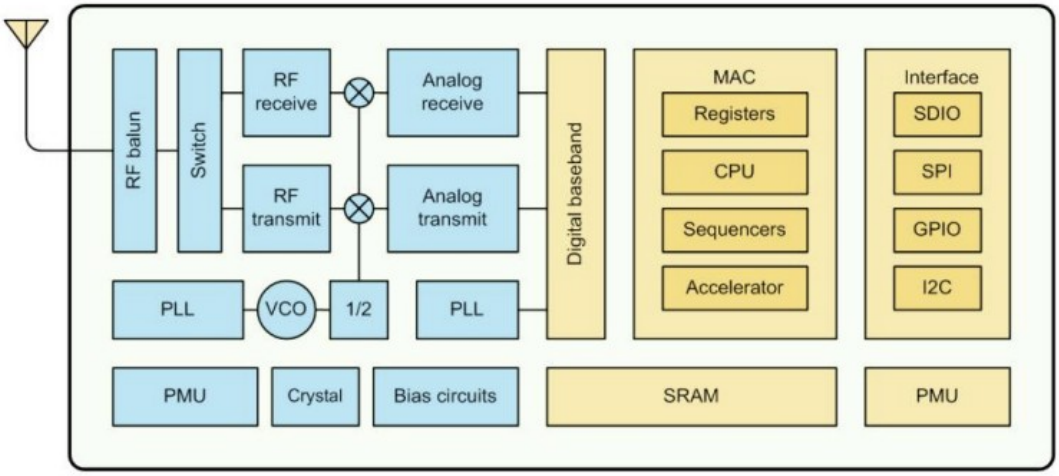
Per fare in modo che il nostro PC risponda, Probabilmente bisognerà disattivare il firewall o creare un'eccezione per consentire il passaggio della richiesta del server.

In pratica è quello che avviene ogni qualvolta digitiamo l'indirizzo di una pagina web, ed in questo caso la pagina web si trova in una determinata posizione all'interno del PC dove viene installato XAMPP.

Ma non sempre abbiamo un PC con la possibilità di installare questi strumenti, come ad esempio nel caso in cui abbiamo una scheda a microcontrollore. Nello specifico andiamo a vedere come installare un webserver sulla scheda ESP8266, e più esattamente prendiamo in esame la scheda Wemos D1 mini, dotata di un ESP8266 ESP-12F.



La scheda in questione ha 9 pin digitali (GPIO) un pin analogico ed una UART interna (TX e RX).
Ha inoltre un interfaccia con un convertitore seriale USB CH340, collegata al PC viene vista come una periferica seriale COM (come avviene con Arduino) se non viene vista dal PC occorre installare i driver del convertitore seriale-USB.
La scheda opera ad una tensione di 3,3Volt, pertanto bisogna utilizzare dei convertitori di tensione se la si vuol collegare a dispositivi che operano a 5V.
All'interno del modulo troviamo l'hardware per la comunicazione SPI e I2C e di un convertitore ADC a 10 bit.



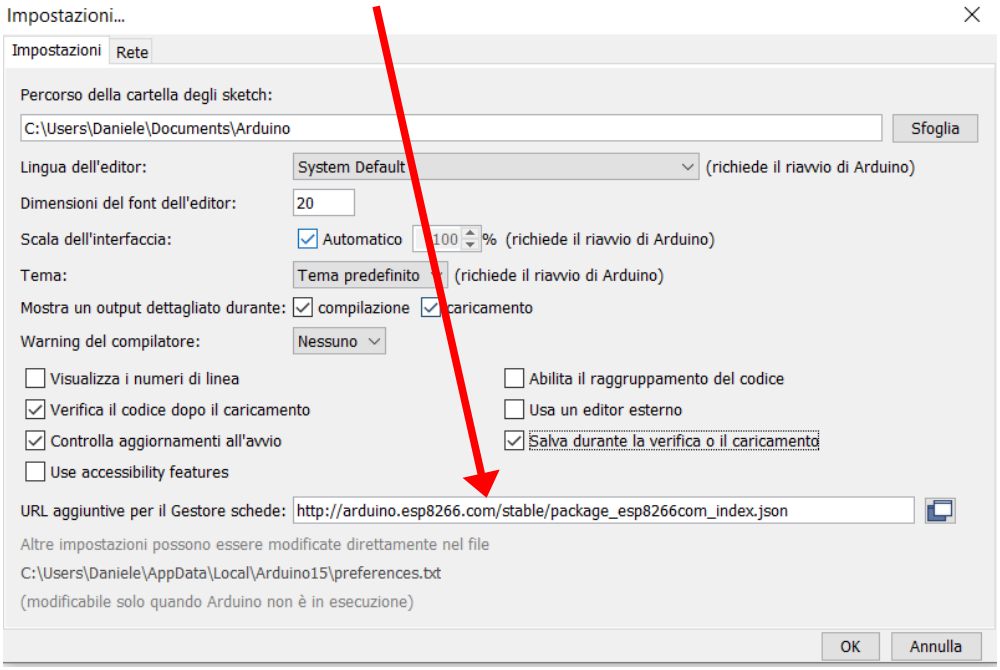
schema a blocchi interno all'ESP8266

La scheda è dotata di una CPU Extensa Tensilica L106 a 32 bit che opera alla frequenza di 80MHz, una memoria RAM di 36kB ed una memoria Flash di 4MB.

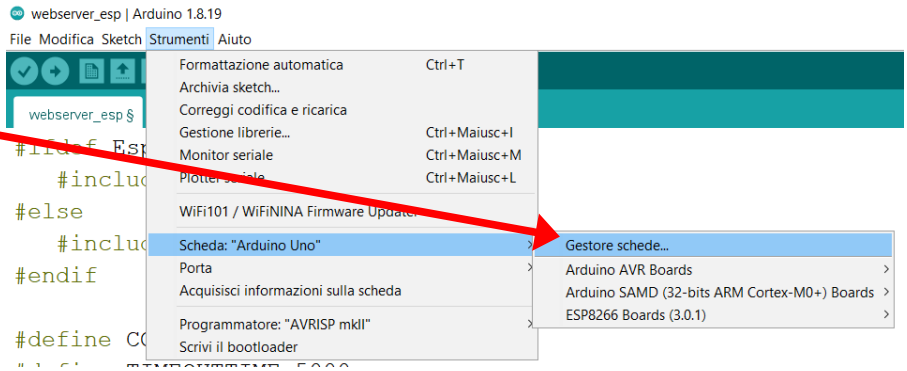
PROGRAMMAZIONE ED AMBIENTE DI SVILUPPO

La scheda può essere programmata con ambiente LUA o anche con l'IDE di Arduino, opzione quest'ultima consigliabile.

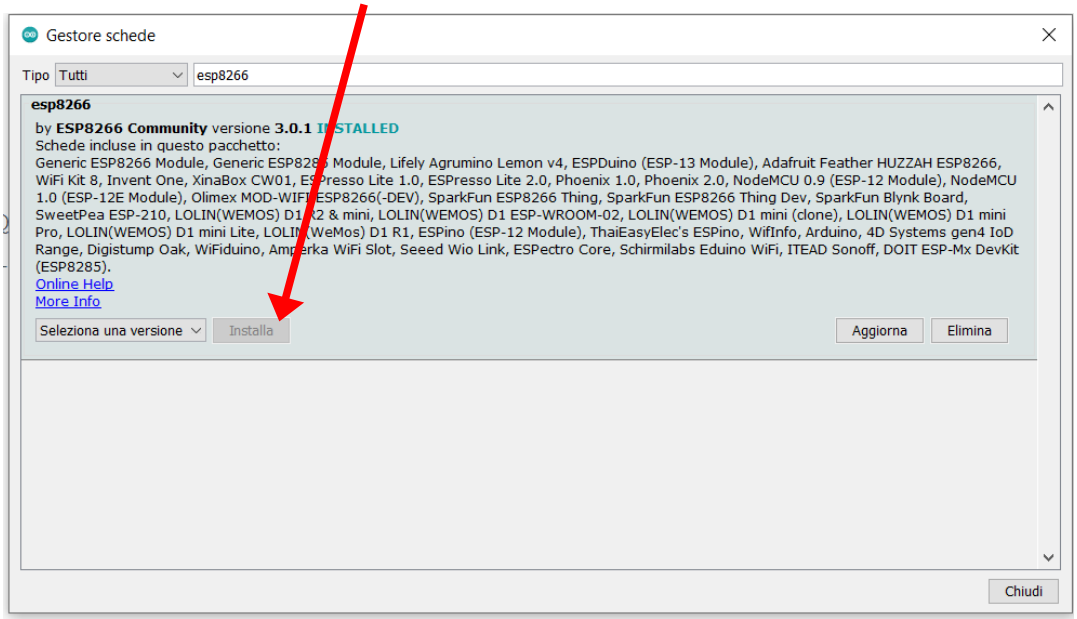
Innazitutto occorre inserire nella sezione "file-impostazioni-URL agiguntive per il Gestore schede" il seguente link: *http://arduino.esp8266.com/stable/package_esp8266com_index.json*



Successivamente occorre installare le librerie dal gestore delle schede di Arduino.

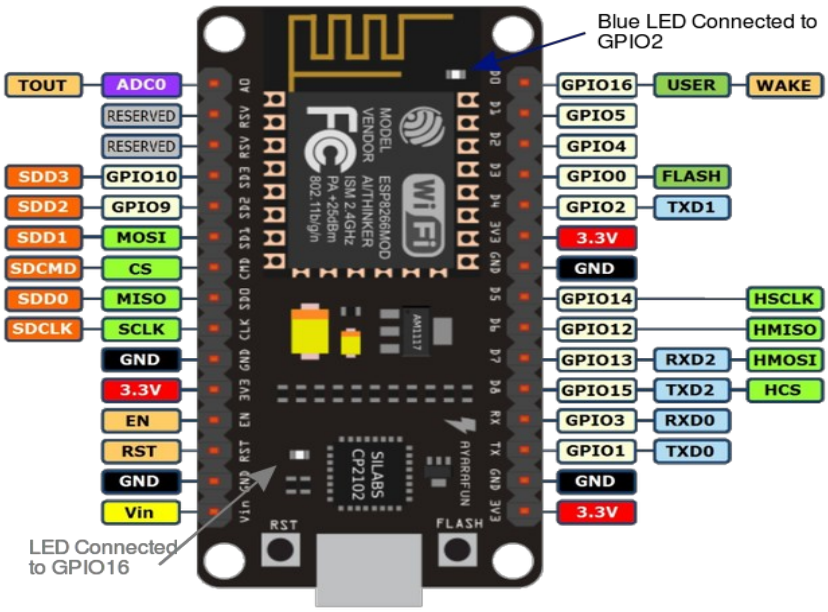


Inserire ESP8266 sulla barra di ricerca ed installare il pacchetto che verrà fuori dalla ricerca.

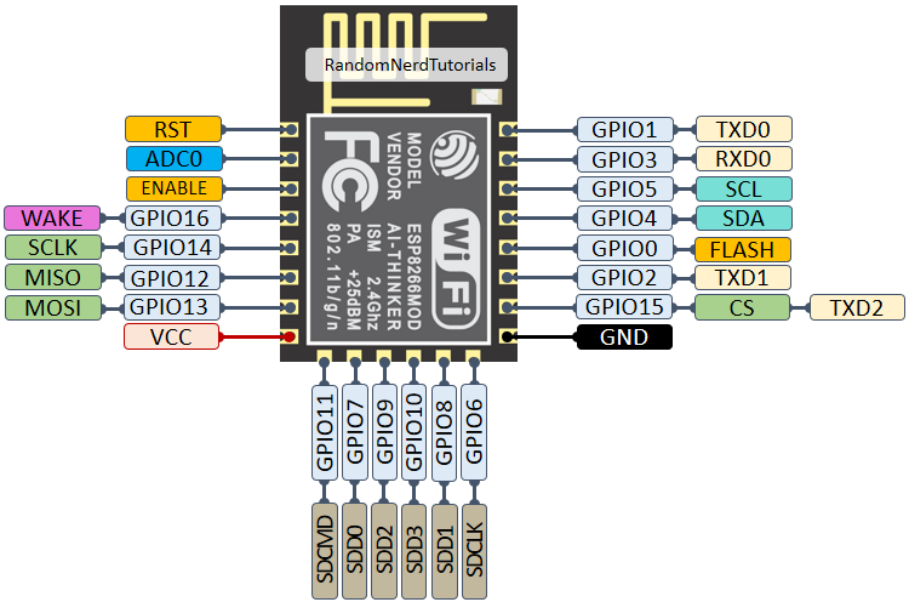


A questo punto la nostra IDE di Arduino sarà pronta per gestire le schede con ESP8266 come la D1 Mini o anche altre schede che montano lo stesso microcontrollore ESP, come ad esempio la NODEMCU.
Anche questa scheda può essere programmata con l'IDE di Arduino, facendo attenzione a scegliere la scheda giusta nell'apposita sezione: *"Strumenti"* – *"NodeMCU 0.9 oppure NodeMCU 1.0"*

Questa scheda monta lo stesso microcontrollore della D1 Mini, ma presenta delle differenze in quanto utilizza altri pin del microcontrollore ESP8266, non utilizzati invece nella scheda D1 Mini.

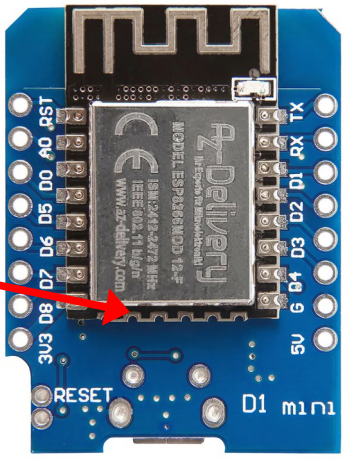


L'ESP8266 ha la seguente piedinatura.

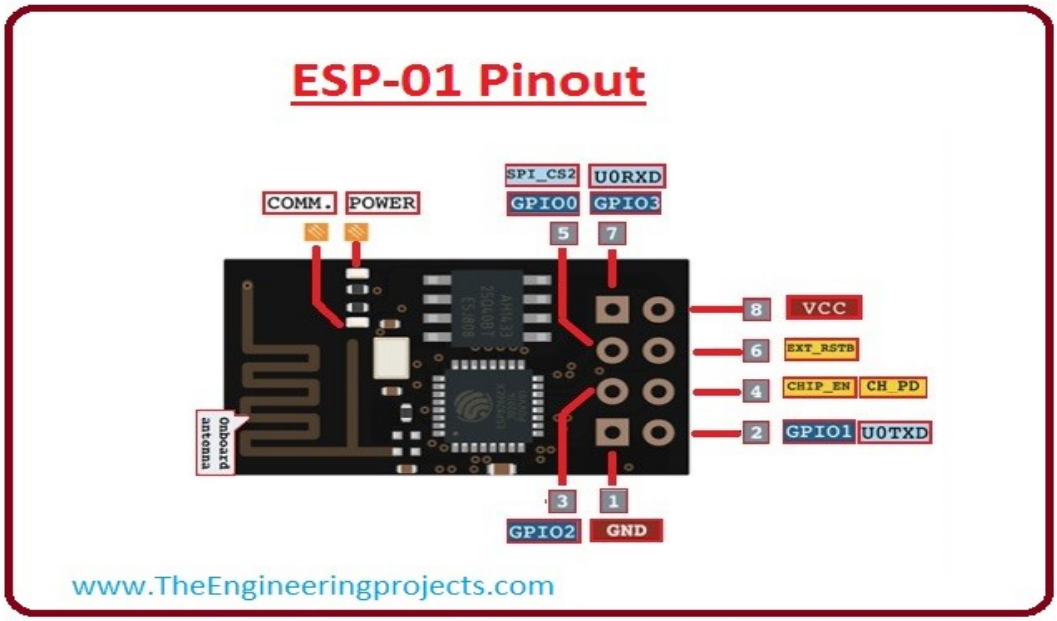


Nel caso della D1 Mini i GPIO centrali non sono collegati.

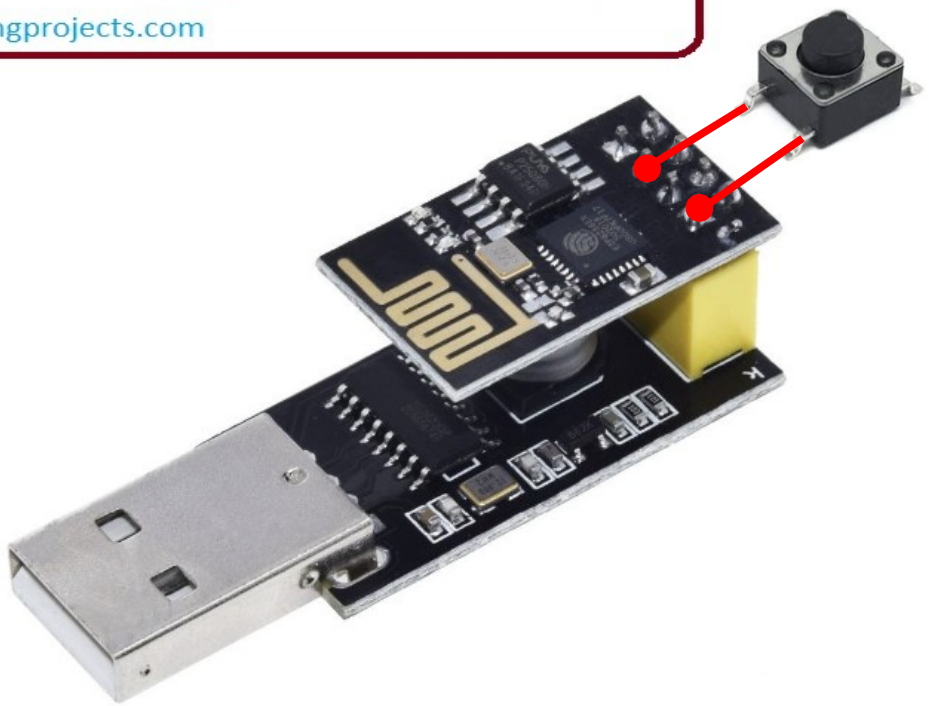
Invece nella scheda NodeMcu vengono utilizzati tutti i pin.



Possiamo anche programmare la scheda ESP01



Utilizzando un adattatore USB-Seriale, posso collegare la scheda alla porta USB e programmarla sempre con l'IDE di Arduino. In questo caso però dovrò aggiungere un pulsante tra il pin GPIO0 e GND per mettere la scheda in modalità programmazione.



La scheda infatti normalmente ha un software per decodificare ed eseguire dei comandi AT, per essere programmata pertanto occorre montare un pulsante come detto sopra.

Fatta questa doverosa premessa sulla panoramica di schede che montano il modulo ESP, andiamo a vedere come caricare a bordo di una di queste schede il WEBServer.

Innanzitutto dobbiamo sapere che il modulo ESP può operare in modo STA, n modo AP o in entrambi i modi.

Modo STA.

In questa modalità il modulo si connette ad un router, che assegnerà al modulo un indirizzo IP con il quale potremo raggiungere la scheda.

Modo AP (Access Point)

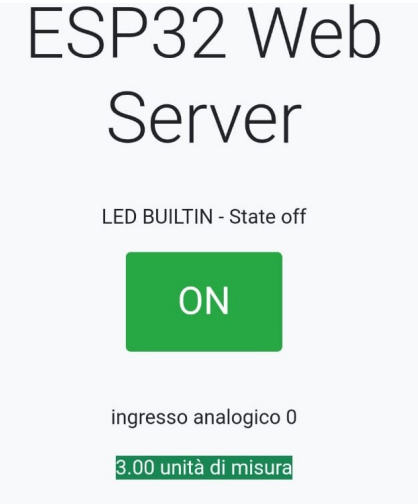
In questa modalità la scheda diventa un piccolo router con una sua rete, che consente però un numero di dispositivi connessi molto limitato.

Caricare a bordo di una scheda ESP un WEBServer, significa fare in modo che alla richiesta HTTP di un client, venga inviata una risposta contenente un codice HTML, con la possibilità di effettuare delle operazioni in caso in cui il client esegua una richiesta con il modo GET previsto dal protocollo HTTP.

Perciò la scheda dopo essersi connessa ad una rete WIFI, rimarrà in ascolto di eventuali richieste di connessione da parte di un Client, come accade con il demone descritto prima in ascolto sulla porta 80.

Analizziamo ora un programma di esempio che non farà altro che consentirci di gestire un led ed un ingresso analogico.

Il risultato del codice riportato sotto è la pagina web di fianco, dove c'è la possibilità di inviare un comando, e ricevere lo stato di un bit o il valore di una variabile.



webserver_esp §

```
#ifdef Esp32
   #include <WiFi.h>
#else
   #include<ESP8266WiFi.h>
#endif

#define CONNECT_TIME 30000
#define TIMEOUTTIME 5000

const char* ssid = "Nome rete wifi";
const char* password = "qwerty";
WiFiServer server(80);

String header;
String LEDState = "off";

size_t currentTime = millis();
size_t previousTime = 0;
```

CONNECT_TIME *sono 30 secondi e cioè il tempo massimo di attesa della connessione al Router.*

TIMEOUTTIME *sono 5 secondi ed è il tempo di attesa della richiesta del client.*

ssid *e* ***password*** *sono i dati della rete wifi*

Con WiFiServer creiamo l'oggetto ***server*** *che lavora sulla porta 80*

header *e* ***LEDstate*** *sono due variabili di tipo stringa.*

Con ***size_t*** *dichiariamo due variabili intere la cui dimensione è variabile ed al massimo può essere 4 byte.In currentTime ci va il tempo in millisecondi all'avvio, infatti la funzione millis() restituisce il tempo trascorso dall'avvio del programma.*

```
void setup() {
  Serial.begin(115200);
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, HIGH);
  Serial.print("Attempting to connect to ");
  Serial.println(ssid);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  size_t start_time = millis();
  size_t dot_time = millis();
  while ((WiFi.status() != WL_CONNECTED) && (start_time + CONNECT_TIME) > millis()) {
    if (dot_time + 250 < millis()) {
      Serial.print(".");
      dot_time = millis();
    }
  }
  if (WiFi.status() != WL_CONNECTED){
    Serial.println("Connection failed");
    while (1);
  }
  Serial.println("");
  Serial.println("WiFi connected.");
  Serial.println("ESP32 IP address: ");
  Serial.println(WiFi.localIP());
  server.begin();
}
```

Nel setup descritto sopra viene inizializzata la comunicazione seriale sui monitor seriale alla velocità di 115.200bps. Dopodiché viene definito il LED presente sulla scheda (LED_BUILTIN) come uscita e subito spento con l'istruzione ***digitalWrite(LED_BUILTIN,HIGH)*** *in quanto il led è connesso verso VCC, perciò si accende con un livello basso e si spegne con un livello alto.*

I vari Serial.print o println, scrivono sul monitor seriale le informazioni relative all'esecuzione del programma.

Con ***WiFi.mode(WIFI_STA)*** *si attiva la modalita Station sul modulo ESP che consente la connessione ad una rete wifi esistente con ssid e password tramite la funzione* ***WiFi.begin****.*

Nelle due variabili ***start_time*** *e* ***dot_time*** *viene salvato il tempo in millisecondi dall'avvio del programma.*

Successivamente nel ciclo while si attende la restituzione della risposta ***WL_CONNECTED*** *dalla funzione* ***WiFi.Status()*** *entro il tempo previsto dalla variabile* ***CONNECT_TIME. I*** *possibili valori restituiti dalla funzione WIFI.Status() sono disponibili sulla guida di Arduino presente nel menu dell'IDE, sotto la sezione librerie scegliendo la libreria WiFi.*

L'eventuale connessione viene poi segnalata sul monitor seriale insieme all'indirizzo IP ottenuto dalla funzione ***WiFi.localIP()****. Una volta effettuata la connessione si inizializza il server con server.begin(), in questo modo viene attivato l'ascolto sulla porta 80 dei pacchetti HTTP.*

```
void loop() {
  WiFiClient client = server.available();
  if (client) {
    currentTime = millis();
    previousTime = currentTime;
    Serial.println("New Client.");
    String currentLine = "";

    while (client.connected() && currentTime - previousTime <= TIMEOUTTIME) {
      currentTime = millis();
      if (client.available()) {
        char c = client.read();
        Serial.write(c);
        header += c;

        if (c == '\n') {
          if (currentLine.length() == 0) {
            Serial.println("Invio pagina WEB");
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();
```

*All'inizio del loop, con **WiFiClient** viene creato l'oggetto client che assume il valore di ritorno della funzione **server.available()** che ci restituisce un TRUE se c'è un client connesso.*

*Se c'è un client connesso viene preso il tempo con **millis()** e fino a quando il client è connesso (ciclo while(client.connected()….) si testa se c'è un carattere disponibile con **client.available()** e successivamente il carattere viene letto e concatenato nella stringa **header.***

*La richiesta HTTP termina con due caratteri **/n**, al termine della stringa viene inviata la pagina HTML con una serie di **client.println()**.*

Richiesta http iniziale della pagina caratteri ASCII ricevuti.

```
GET / HTTP/1.1
Host: 192.168.43.119
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://192.168.43.119/
Accept-Language: it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7
Accept-Encoding: gzip, deflate
```

Richiesta http iniziale della pagina valori decimali ricevuti.

Come si può notare nella richiesta sopra, sono state inviate 10 righe e possiamo notare anche analizzando i valori decimali (010=/n cioè nuova linea).

```
071 069 084 032 047 032 072 084 084 080 047 049 046 049
010 072 111 115 116 058 032 049 057 050 046 049 054 056 046 052 051 046 049 049 057
010 067 111 110 110 101 099 116 105 111 110 058 032 107 101 101 112 045 097 108 105 118 101
010 067 097 099 104 101 045 067 111 110 116 114 111 108 058 032 109 097 120 045 097 103 101 061 048
010 085 112 103 114 097 100 101 045 073 110 115 101 099 117 114 101 045 082 101 113 117 101 115 116 115 058 032 049
010 085 115 101 114 045 065 103 101 110 116 058 032 077 111 122 105 108 108 097 047 053 046 048 032 040 087 105 110
    100 111 119 115 032 078 084 032 049 048 046 048 059 032 087 105 110 054 052 059 032 120 054 052 041 032 065 112
    112 108 101 087 101 098 075 105 116 047 053 051 055 046 051 054 032 040 075 072 084 077 076 044 032 108 105 107
    101 032 071 101 099 107 111 041 032 067 104 114 111 109 101 047 049 048 057 046 048 046 048 046 048 032 083 097
    102 097 114 105 047 053 051 055 046 051 054
010 065 099 099 101 112 116 058 032 116 101 120 116 047 104 116 109 108 044 097 112 112 108 105 099 097 116 105 111
    110 047 120 104 116 109 108 043 120 109 108 044 097 112 112 108 105 099 097 116 105 111 110 047 120 109 108 059
    113 061 048 046 057 044 105 109 097 103 101 047 097 118 105 102 044 105 109 097 103 101 047 119 101 098 112 044
    105 109 097 103 101 047 097 112 110 103 044 042 047 042 059 113 061 048 046 056 044 097 112 112 108 105 099 097
    116 105 111 110 047 115 105 103 110 101 100 045 101 120 099 104 097 110 103 101 059 118 061 098 051 059 113 061
    048 046 057
010 082 101 102 101 114 101 114 058 032 104 116 116 112 058 047 047 049 057 050 046 049 054 056 046 052 051 046 049
    049 057 047
010 065 099 099 101 112 116 045 076 097 110 103 117 097 103 101 058 032 105 116 045 073 084 044 105 116 059 113 061
    048 046 057 044 101 110 045 085 083 059 113 061 048 046 056 044 101 110 059 113 061 048 046 055
010 065 099 099 101 112 116 045 069 110 099 111 100 105 110 103 058 032 103 122 105 112 044 032 100 101 102 108 097
    116 101
010
010
```

Al termine della richiesta ci sono due 010 cioè viene inviato per due volte il carattere Ascii /n.

La stringa **currentLine**, contiene l'intero messaggio e viene svuotata una volta che si ottiene la ricezione di due caratteri /n. Il carattere /r non viene aggiunto alla stringa header per la condizione **else if (c != '\r').**

All'interno del ciclo while, ci sono due if consecutivi, il primo analizza il carattere /n il secondo invece verifica lo svuotamento della stringa currentLine. Al verificarsi di queste due condizioni viene inviata la pagina web, e viene analizzato il contenuto della richiesta GET per vedere se c'è qualche comando da eseguire.

```
            if (header.indexOf("GET /LED/on") >= 0) {
              Serial.println("LED  on");
              LEDState = "on";
              digitalWrite(LED_BUILTIN, LOW);
            } else if (header.indexOf("GET /LED/off") >= 0) {
              Serial.println("LED off");
              LEDState = "off";
              digitalWrite(LED_BUILTIN, HIGH);
            }
            client.println("<
            client.println("<
            client.println("<
            client.println("<
            client.println("<
            client.println("<
            client.println("<
              double lettura = analogRead(0)/1.0 ;
              String valore = String(lettura);
              client.println("<p class=\"par\">ingresso analogico 0 </p>");
              if ( lettura < 100)
                client.println("<p> <span class=\"ingresso analogico\" style
              else if ( lettura > 100)
                client.println("<
              else
                client.println("<
              client.println("<br
              client.println("</b
              client.println();
              break;
            } else {
              currentLine = "";
            }
          } else if (c != '\r') {
            currentLine += c;
          }
        }
      }
      header = "";
      client.stop();
      Serial.println("Client disconnected.");
      Serial.println("");
    }
    delay(1);
}
```

Inserire qui la pagina HTML, inviando ogni riga con il client.println(...)

In questa parte viene concatenato alla stringa contenente il codice html, il valore letto precedentemente convertito in stringa

Con il metodo **(header.indexOf("GET /LED/on") >= 0)** si va a ricercare nella stringa ricevuta il comando inviato con il metodo GET e si esegue l'operazione corrispondente.

Informazioni sul metodo indexOf() al seguente link:

https://www.arduino.cc/reference/en/language/variables/data-types/string/functions/indexof/

con **String valore = String(lettura);** si trasforma il dato presente su lettura in una stringa chiamata valore, che verrà concatenata nel codice HTML ed inviata tramite **client.println().**

Il programma può ovviamente essere strutturato in maniera diversa, l'importante è comprendere il funzionato dell'esempio sopra descritto ed allegato alla dispensa in maniera integrale.

Codice sorgente:

https://danielepostacchini.it/wp-content/uploads/2023/02/webserver_esp.zip

Video WebServer:

https://youtu.be/Wn1cK8TlGu0