# Comunicazioni con Arduino

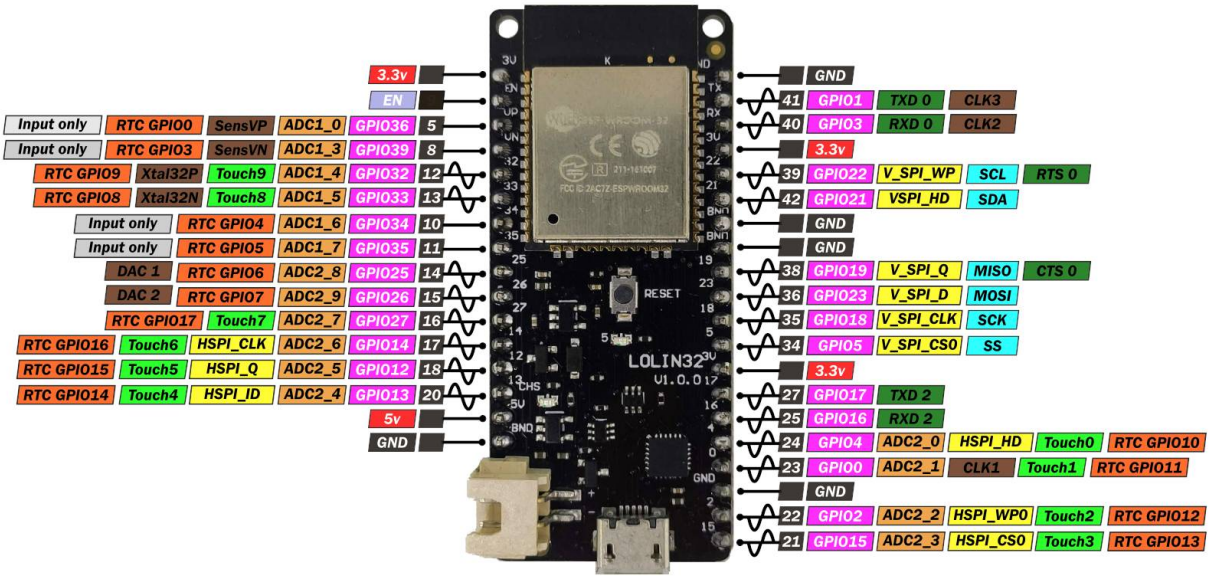
In questa traccia si sperimentano alcune tecniche di comunicazione che si possono utilizzare per mettere un microcontroller in comunicazione con altri dispositivi.
In particolare si sperimentano due tipologie di comunicazione:

- Comunicazione punto a punto con Bluetooth
- Comunicazione di rete TCP/IP con WiFi

**Wemos Lolin32**

Per lo sviluppo di queste applicazioni si usa il dispositivo Wemos Lolin32 che è Arduino-compatibile basato sul SoC (System on Chip) ESP32 che integra un microcontroller Xtensa LX6 con le periferiche BLE e WiFi.

**Pinout del Wemos Lolin32**

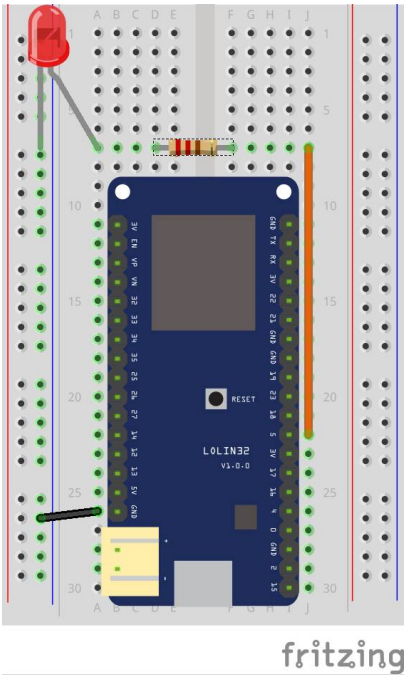



**NOTE IMPORTANTI**

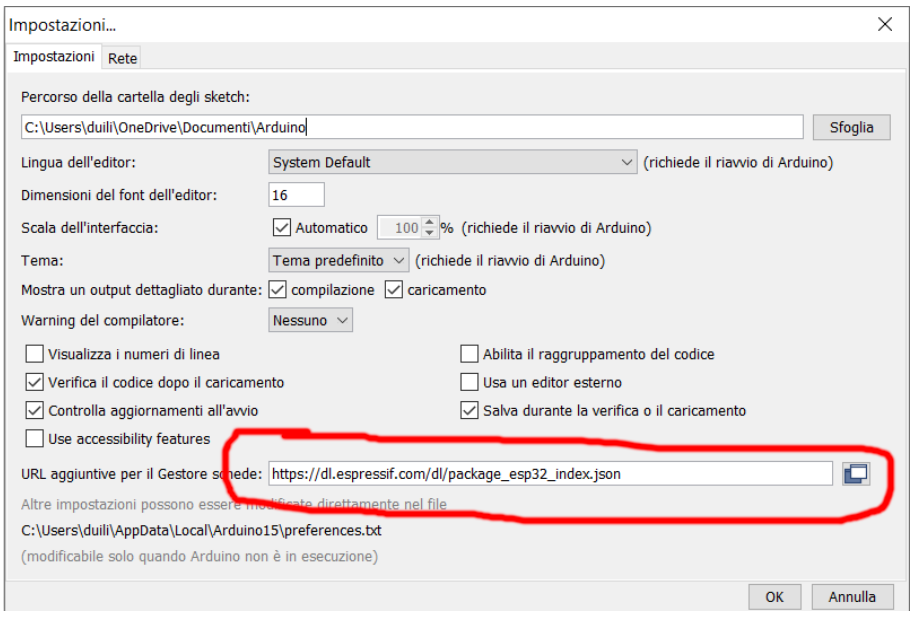
- Wemos Lolin32 è software compatibile con Arduino ma NON hardware compatibile. In particolare:
- Il dispositivo può essere alimenta a 5V via USB oppure a 3.7V via JST (batteria LiPo) ma opera comunque a 3.3V
- Se alimentato a 5V espone tale tensione sul pin 5V e può essere usata per alimentare i dispositivi a 5V come ad esempio il display
- Se alimentato a 3.7V non espone i 5V sul pin 5V. Se si vuole usare una batteria LiPo ma servono i 5V è necessario interporre un convertitore da 3.7V a 5V tra batteria e Wemos alimentandolo via USB.
- Tutti i segnali anologici e digitali operano a 3.3V
- Non tutti i pin di ingresso sono dotati del pullup interno. Nel dubbio meglio inserire una resistenza esterna di pullup
- Alcuni pin digitali funzionano al tocco; basta quindi collegarli ad una superficie metallica.
- Tutti i pin di uscita possono essere usati come PWM
- Il convertitore A/D è a 12 bit quindi il suo range 0-4095
- Bluetooth e WiFi non possono essere usati contemporaneamente ma è possibile fare una commutazione runtime tra le due periferiche

**Comunicazione punto a punto via Bluetooth**
Il Wemos Lolin32 è dotato di una periferica BLE che consente la realizzazione di una comunicazione client/server ma in questo esempio verrà realizzata una comunicazione punto a punto come quella di un Bluetooth classico.
L'obiettivo del progetto è comandare l'accensione di un LED e conoscere lo stato del LED attraverso uno smartphone.

**Schema di montaggio del Wemos**




**Configurazione dell'IDE di Arduino**
La scheda Wemos non è nativamente presente nell'IDE di Arduino.
Per consentire l'utilizzo della scheda si deve accedere al menu File > Impostazioni … ed inserire nella casella "URL aggiuntive per il Gestore schede" l'URL:
**https://dl.espressif.com/dl/package_esp32_index.json**



Accedendo al menu Strumenti > Scheda > Gestore Schede …
Si apre la finestra di Gestione Schede; inserendo nella casella di ricerca la chiave esp32 viene trovato il chip ESP32 che può essere installato.
Completata l'installazione dal menu Strumenti > Scheda sarà possibile selezionare il gruppo ESP32 Arduino contenente tra l'altro la scheda **Wemos Lolin32**.

**Terminale di collaudo**
L'obiettivo del progetto è di comunicare con una app dedicata su smartphone, tuttavia quando si sviluppa un progetto di comunicazione che coinvolge due software di comunicazione non è opportuno sviluppare contemporaneamente su entrambi i dispositivi perché in presenza di un errore è difficile capire da quale parte sia.
Prima di sviluppare l'app dedicata per smartphone conviene quindi usare un software di collaudo.
In questo caso si può utilizzare un Terminale Bluetooth.
Esistono molte app Terminale Bluetooth sia per Android che per Iphone.
In questa traccia gli esempi si basano su "Bluetooth Terminal 1.2 Juan Sebastian Ochoa Zambrano" per Android ma si può usare un qualsiasi terminale.
https://play.google.com/store/apps/details?id=ptah.apps.bluetoothterminal&hl=it
Un terminale bluetooth si connette al dispositivo remoto a patto che sia stato precedentemente associato ed invia o riceve stringhe di testo.
Per poter fare l'associazione del BT sullo smartphone è necessario che il firmware del microcontroller abbia inizializzato il BT.

**Esercizio "Serial to SerialBT"**
In questo esercizio si testa una semplice comunicazione testuale bidirezionale tra il terminale bluetooth dello smartphone e il monitor seriale su PC del microcontroller.

**Firmware del microcontroller**
Dopo aver impostato la scheda su Wemos Lolin32 è necessario includere nel progetto la libreria "BluetoothSerial" che consente di usare il BT come un canale seriale.

```
#include "BluetoothSerial.h"
```

Si dichiara un oggetto di nome SerialBT di classe BluetoothSerial.

```
BluetoothSerial SerialBT;
```

Nel setup si inizializzano sia la seriale del monitor (porta USB del microcontroller) che la seriale emulata dal BT:

```
Serial.begin(115200);
SerialBT.begin("ESP32xx"); //Bluetooth device name
```

N.B.
La stringa inserita nell'argomento del metodo begin sarà il nome del device esposto per l'associazione. Per evitare ambiguità è opportuno che ogni gruppo utilizzi un nome diverso dagli altri, ad esempio con numeri progressivi: ESP3201, ESP3202, ...
Nel loop vengono testati entrambi i canali per verificare la presenza di caratteri in attesa di essere gestiti e ogni carattere presente in entrata da un canale viene trasferito in uscita sull'altro canale.
In questo modo le stringhe inserite nello smartphone vengono visualizzate sul monitor e viceversa.

```
if (Serial.available()) {          //legge da monitor ed invia a BT
  SerialBT.write(Serial.read());
}
if (SerialBT.available()) {        //legge da BT ed invia a monitor
  Serial.write(SerialBT.read());
}
```

**N.B.**

- I metodi read() di entrambe le classi sono sospensivi quindi se invocati quando il buffer di ricezione è vuoto provocano la sospensione del loop. Per evitare questo evento che, in caso di un canale vuoto impedirebbe di leggere l'altro anche se contenente dati entrambe le read() sono condizionate alla verità del metodo available() che NON è sospensivo e verifica se ci sono caratteri nel buffer di ricezione; in questo modo la read() viene eseguita solo se ci sono caratteri ancora da leggere.
- Ogni carattere letto da un canale viene inoltrato all'altro canale
- Sia il monitor che il terminale non inviano i caratteri uno alla volta ma tutti insieme quando si preme il bottone Send
- Il monitor, a differenza del terminale aggiunge un carattere "caporiga" (carattere non stampabile NewLine=0x0A) quindi i messaggi nel terminale compaiono sempre in righe separate mentre nel monitor tutti sulla stessa riga. Per avere messaggi a caporiga nel monitor è necessario digitare il tasto "caporiga" prima di premere Send. Se al contrario si vuole togliere il caporiga dal terminale si deve modificare la configurazione in basso a destra del monitor da "A capo (NL)" a "Nessun fine riga".

**Associazione dei dispositivi**

Per poter comunicare i due dispositivi devono essere associati.

Il microcontroller diventa disponibile per l'associazione quando viene eseguito un firmware contenente il metodo begin con un nome dispositivo.

Da quel momento uno smartphone che faccia una ricerca dei dispositivi BT vicini vede il nome dispositivo e lo può associare.

Si deve abilitare il bluetooth sullo smartphone ed avviare la ricerca di dispositivi vicini.

**Dispositivo associabile** **Associazione dispositivo** **Dispositivo associato**



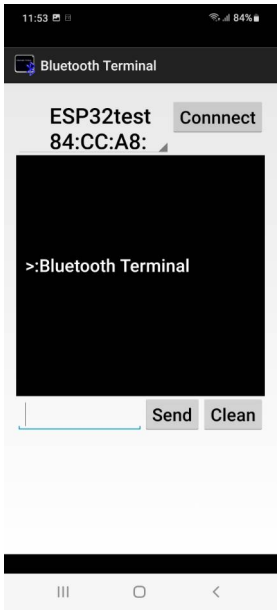

Dopo l'associazione è possibile connettersi al dispositivo con il Bluetooth terminal

**Connessione**

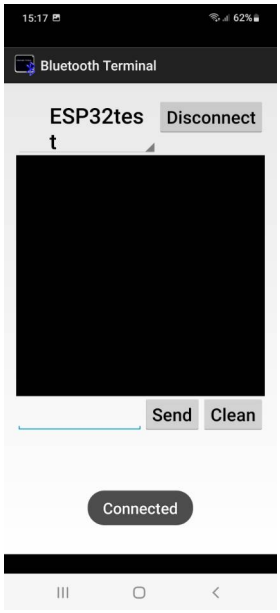
Aprendo l'app bluetooth terminal sullo smartphone, con bluetooth attivato, si vede una casella a discesa contenente tutti i dispositivi associati.

Si seleziona il dispositivo e si preme il bottone Connect.Se la connessione ha successo compare brevemente  il messaggio Connected ed il bottone diventa Disconnect.

**Connessione**




**Dispositivo connesso**

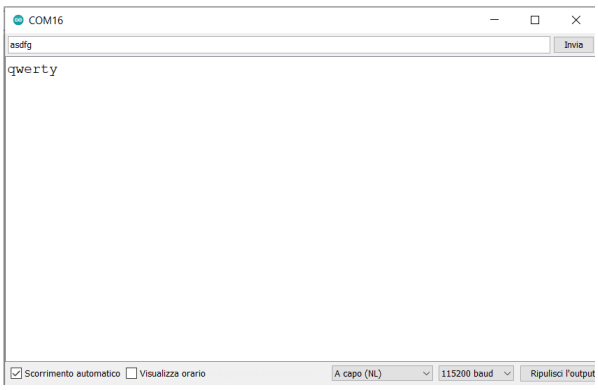



**Scambio di messaggi testuali**

Da questo momento è possibile comunicare in entrambe le direzioni scrivendo messaggi nel terminale e nel monitor ed inviando con il bottone Send.

**Lato smartphone:**
**invia qwerty<nl>  e riceve asdfg<nl>**


**Lato controller:**
**invia asdfg<nl>  e riceve qwerty<nl>**




**Accensione di un LED con comando da smartphone**

La struttura del firmware è analoga al caso precedente.

La differenza rispetto al primo test è che in questo caso il firmware non deve inoltrare i messaggi ricevuti come sono ma li deve interpretare.

Supponiamo che dal terminale possano arrivare solo due messaggi ON ed OFF.
Per agevolare il riconoscimento del messaggio e intercettare eventuali errori di digitazione o di comunicazione conviene definire un "terminatore di messaggio" cioè un carattere speciale che non può comparire all'interno del messaggio e che certifichi che il messaggio è terminato.
Solo a questo punto il firmware analizza il messaggio e se lo riconosce attua il comando altrimenti lo ignora. Si può utilizzare il NewLine già visto come terminatore.
I due comandi quindi sono:
**ON<nl>**
**OFF<nl>**
dove <nl> è un metatermine che sta ad indicare il codice ASCII esadecimale 0x0A.
Il loop deve quindi estrarre i caratteri dalla seriale BT ed accumularli in una stringa fino al riconoscimento del terminatore.
Raggiunto il terminatore la stringa viene interpretata controllando se è una delle due stringhe valide e in questo caso si attua il LED e si restituisce sulla seriale bluetooth lo stato del LED altrimenti la stringa viene scartata.
In ogni caso la stringa viene svuotata per avviare una nuova interpretazione.
Per la gestione della stringa si può utilizzare sia una string "C" (array di char terminato da un NULL) sia un oggetto C++ di classe String. L'esempio si riferisce a questo secondo caso che semplifica notevolmente il codice.
Si dichiara uno oggetto rcvBuffer di classe String

```
String rcvBuffer;
```

questo tipo di dichiarazione produce una stringa vuota.
Nel setup oltre alla inzializzazione del canale BT si inizializza lo stato del LED a spento:

```
pinMode(LED_BUILTIN,OUTPUT);
digitalWrite(LED_BUILTIN,LOW);
```

il pin GPIO5 scelto nel montaggio corrisponde al pin 5 di Wemos Lolin32 e al LED interno blu. Tuttavia il led interno blu funziona in logica negativa quindi è acceso con LOW  e spento con HIGH mentre il LED esterno è in logica positiva.
Nel loop ogni volta che un carattere è disponibile sul canale BT lo si appende in fondo alla stringa a meno che non sia il carattere 0x0A. Se è il carattere 0x0A invece si controlla se la stringa è uno dei due comandi validi e in tale caso si attua il comando e si manda al terminale lo stato del LED. Dopo ogni attuazione la stringa va vuotata per consentire una nuova acquisizione.

```
if (SerialBT.available()) {              //se c'è un char in arrivo
  char rcvChar=SerialBT.read();          //legge il char
  if(rcvChar!=0xA) {                     //non è il terminatore
    rcvBuffer+=rcvChar;                  //appende nella stringa
  }
  else {                                 //è il terminatore
    if(rcvBuffer=="ON") {                //comando ON
      digitalWrite(LED_BUILTIN,HIGH);    //accende LED
        SerialBT.println("HIGH");        //stato a smartphone
    }
    if(rcvBuffer=="OFF") {               //comando OFF
      digitalWrite(LED_BUILTIN,LOW);     //spegne LED
      SerialBT.println("LOW");           //stato a smartphone
    }
    rcvBuffer="";                        //svuota il buffer
 }
}
```

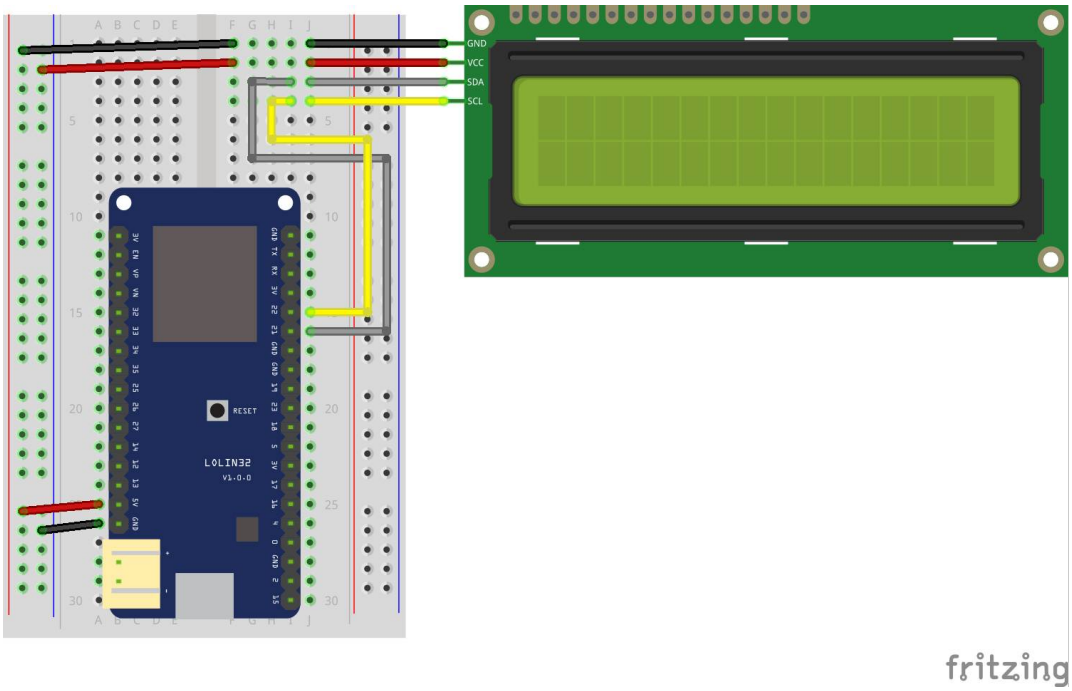
**Client HTTP**

Il Wemos Lolin32 è dotato di una periferica WiFi che consente la connessione ad un access point e quindi la connessione alla rete Internet.

L'obiettivo del progetto è connettere il dispositivo alla rete di istituto ed effettuare una richiesta di tipo GET ad un server HTTP. Quello che viene restituito non è una pagina web ma una stringa JSON contenente informazioni per il dispositivo.

Nell'esempio l'informazione è la data/ora che viene mostrata sul display.

**Schema di montaggio**




**N.B.**

- Con Wemos non è possibile utilizzare la scheda grove per Arduino UNO quindi per utilizzare il display Grove è necessario inserire quattro ponticelli nel connettore del display e collegarli alla breadboard
- Il display Grove funziona solo a 5V quindi il pin di alimentazione va collegato al pin 5V
- I pin SDA ed SCL del display vanno collegati ai corrispondenti pin di Wemos che si trovano rispettivamente sui pin 21 e 22
- Si può testare la funzionalità del cablaggio con il precedente esercizio sul display.

**Verifica del servizio remoto**

Per sincronizzare la propria data ed ora il controller deve effettuare una richiesta HTTP ad un servizio remoto che restituisce una stringa JSON.

Prima di partire con lo sviluppo del firmware conviene controllare l'accessibilità al servizio visitandolo con un browser.

L'URL è: http://www.schoolmakerday.it/logger/time.php

e dovrebbe restituire una stringa JSON del tipo:

```
{"status":"OK","data":{"year":"2022","month":"03",
"day":"16","hour":"16","minute":"44","second":"16"}}
```

**Codice dello script di servizio remoto**

```
<?php
$time=explode('-',date('Y-m-d-H-i-s'));
$t=[
    'year'=>$time[0],
    'month'=>$time[1],
    'day'=>$time[2],
    'hour'=>$time[3],
    'minute'=>$time[4],
    'second'=>$time[5],
];

$response=[
    'status'=>'OK',
    'data'=>$t,
];

//output
header("Content-Type: application/json;charset=utf-8");
echo json_encode($response);
```

**Connessione all'access point**

Il primo passo di sviluppo è la connessione all'access point.
La periferica WiFi di ESP32 ha molti modi di funzionamento (Station, Access Point, Bridge).
In questo esempio si usa il modo Station in cui il dispositivo si connette ad un access point e chiede un numero IP della rete in cui l'access point è inserito.
Nel primo test ci si limita ad accedere all'access point e a verificare che la connessione abbia successo e quale numero IP è stato assegnato.
Come nel caso di Serial la libreria WiFi.h espone un oggetto di tipo early-binding di nome Wifi che quindi non va istanziato ma se ne usano direttamente le classi.
Data la natura di questo test il codice si trova tutto nel setup ed il loop è vuoto.
Quindi si include la libreria WiFi

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

Nel setup, dopo aver inizializzato Serial per mostrare la diagnostica, si inizializza la WiFi in modo Station e si tenta la connessione con SSID e password dell'access point:

```
WiFi.mode(WIFI_STA);
WiFi.begin("SSID", "password");
```

Al posto di SSID e password mettere gli effettivi valori della rete in cui si vuole inserire il dispositivo
La connessione ad un Access Point non è una operazione istantanea quindi se si testa la connessione immediatamente dopo il begin potrebbe fallire.
E' opportuno attendere in setup fino a quando la connessione non ha avuto successo (potrebbe richiedere alcuni secondi:

```
while (WiFi.status() != WL_CONNECTED) {
  Serial.print('.');
  delay(1000);
}
```

Quando la connessione ha avuto successo è possibile individuare il numero IP assegnato dal DHCP:

```
Serial.println(WiFi.localIP());
```

**Richiesta del servizio remoto**

Il primo passo è creare un client HTTP, effettuare una richiesta di GET per l'indirizzo desiderato e, ottenuta la risposta, fare una diagnosta sul monitor verificando quali dati sono arrivati.

Per creare un client HTTP si deve includere la libreria HTTPClient.h. Non esiste un oggetto predefinito HTTPClient quindi tale oggetto va creato.

```
#include <HTTPClient.h>
HTTPClient http;
```

In una richiesta di GET si deve prima specificare l'URI che si vuole ottenere e poi effettuare la richiesta.

Il server risponde con un codice numerico. Se il numero è negativo si tratta di un errore di connessione mentre i codici positivi (da 1xx a 5xx ) sono risposte del server. Solo la risposta 200 corrisponde ad una risposta valida.

Nel loop si può inserire una richiesta di GET al servizio remoto. Si attende la risposta e si valuta il codice di risposta. Se il codice di risposta è 200 si estrae il contenuto testuale della risposta (payload) e lo si manda al monitor per la diagnostica.

Trattandosi di un time con risoluzione del secondo si può ripetere al richiesta ogni secondo.

Se la richiesta va in errore si riporta il codice di errore.

```
Serial.println("[HTTP] begin...");
http.begin("http://www.schoolmakerday.it/logger/time.php");
Serial.println("[HTTP] GET...");
int httpCode = http.GET();
if(httpCode > 0) {  //connessione ok
  Serial.print("[HTTP] GET... code: ");
  Serial.println(httpCode);
  if(httpCode == HTTP_CODE_OK) { //codice 200: stampa payload
    String payload = http.getString();
    Serial.println(payload);
  }
  else {  //stampa codice di errore 1xx - 5xx
     Serial.print("[HTTP] GET... failed, error:");
     Serial.println(httpCode);
    }
 }
 else { //stampa codice di errore connessione
   Serial.print("[HTTP] GET... failed, error:");
   Serial.println(httpCode);
 }
 http.end();
 delay(1000);
```

**N.B.**

**L'uso della funzione delay(1000) può essere fatto solo per scopo di test perché impedirebbe il funzionamento di qualsiasi altra attività del microcontroller. Nei casi reali si deve usara la funzione millis() introducento una attesa non sospensiva.**

**Interpretazione e presentazione dei dati**

L'estrazione della risposta rende disponibile una stringa di testo il cui formato dipende dall'applicazione.

E' quindi necessario interpretare la stringa per portarla nel formato adatto per l'uso da parte dell'applicazione.

Nell'esempio la stringa ha un formato JSON; conviene quindi utilizzare una libreria JSON per estrarre le varie parti e poi riassemblarle in base alle necessità.
La libreria ArduinoJson non fa parte del core dell'IDE e quindi va installata dal menu Sketch > #include libreria > Gestioni librerie.
L'azione fondamentale per interpretare il contenuto di una stringa JSON è la **deserializzazione** che nel caso della libreria ArduinoJson trasforma la stringa in un array associativo multidimensionale.
Ad esempio nel caso della stringa del servizio che stiamo utilizzando deserializzando si ottiene un array associativo che contiene l'elemento ["status"] e l'elemento ["data"]. L'elemento data è a sua volta un array che contiene gli elementi year, month … quindi ogni elemento è accessibile con la doppia indicizzazione ["data"]["year"], ["data"]["month"] …
Questa è una possibile rappresentazione grafica dell'array associativo generato:

| *status* | **OK \| FAIL** | |
|---|---|---|
| *data* | *year* | **####** |
| | *month* | **##** |
| | *day* | **##** |
| | *hour* | **##** |
| | *minute* | **##** |
| | *second* | **##** |

Il metodo di deserializzazione si chiama **deserializeJson()** ed ha il seguente prototipo

```
DeserializeError deserializeJson(JsonDocument &doc, String  input);
```

Il primo parametro è un parametro di uscita e rappresenta l'array contenente i risultati della deserializzazione, il secondo parametro è un parametro di ingresso e rappresenta la stringa da deserializzare. Il valore restituito dalla funzione è un codice di errore di tipo enumerativo (0=nessun errore)
Il JsonDocument può essere di tipo statico o dinamico. In una applicazione per microcontroller conviene utilizzare l'oggetto di tipo statico StaticJsonDocument a causa del limite di memoria heap.
La versione statica del JsonDocument richiede la definizione delle dimensioni dell'oggetto. Questo dimensionamento può essere fatto in fase di compilazione in base alla struttura della stringa:

```
const int capacity = JSON_OBJECT_SIZE(2) + JSON_OBJECT_SIZE(12);
StaticJsonDocument<capacity> datetime;
DeserializationError error;;
```

Va dichiarato anche l'enumerativo che contiene il codice di errore.
A questo punto è possibile trasformare il payload ottenuto dalla richiesta HTTP in una oggetto JSON:

```
error=deserializeJson(datetime,payload);
if(error){
  Serial.print("deserializeJson() failed: ");
  Serial.println(error.c_str());
}
```

Ed estrarre i singoli elementi dall'oggetto sotto forma di stringhe.

```
String sta=datetime["status"];
Serial.println(sta);
String year=datetime["data"]["year"];
Serial.println(year);
String month=datetime["data"]["month"];
Serial.println(month);
String day=datetime["data"]["day"];
Serial.println(day);
```

```
String hour=datetime["data"]["hour"];
Serial.println(hour);
String minute=datetime["data"]["minute"];
Serial.println(minute);
String second=datetime["data"]["second"];
Serial.println(second);
```

Dai singoli elementi si possono ricavare due stringhe da inviare alle due righe del display

```
String date=day+"/"+month+"/"+year;
Serial.println(date);
String time=hour+":"+minute+":"+second;
Serial.println(time);
```

Per visualizzare sul display il dato è necessario aggiungere le librerie Wire ed rgb_lcd, dichiarare un oggetto rgb_lcd ed impostare i colori di sfondo:

```
#include <Wire.h>
#include <rgb_lcd.h>
rgb_lcd lcd;              //crea oggetto lcd
const int colorR = 0xff; //colori di sfondo
const int colorG = 0xff;
const int colorB = 0xff;
```

Nel setup si deve inizializzare il display

```
lcd.begin(16, 2); //inizializza numero di righe e colonne
lcd.setRGB(colorR, colorG, colorB); //inizializza colore di sfondo
```

E nel loop lo si deve aggiornare con le due stringhe generate dal JSON

```
lcd.setCursor(0, 0);     //cursore all'inizio della seconda riga
lcd.print(date);         //stampa la data
lcd.setCursor(0, 1);     //cursore all'inizio della seconda riga
lcd.print(time);         //stampa l'ora
```

Questa versione del firmware chiede continuamente la data ed ora e la stampa sul display. Un possibile versione alternativa invece potrebbe richiedere la data ed ora solo all'avvio del microcontroller per sincronizzarsi all'accensione (se il microcontroller non contiene un RTC con batteria perde la sincronizzazione allo spegnimento) ma poi si calcola autonomamente l'ora incrementando una variabile di stato che contiene il tempo.

Due considerazioni importanti per questa versione:

- La variabile di stato può essere un oggetto strutturato in giorno, mese, anno, ore, minuti, secondi e ogni secondo si incrementano i secondi facendo gli opportuni riporti oppure può essere un intero (UTC). In questo caso la variabile di stato viene sempre incrementata di 1 e la stringa umanamente leggibile si ottiene per conversione quando si deve fare la visualizzazione (si può utilizzare la libreria timezone).
- In una vera applicazione il riconoscimento del passare di un secondo non può essere fatto chiamando una funzione delay(1000) perché questa sospensione pregiudicherebbe ogni altra attività del microcontroller ma verificando lo scadere del secondo con la funzione millis()