Web server WiFi con Arduino

Messaggi da browser web

Prima di procedere alla composizione di un vero web server WiFi su Arduino, è bene sperimentare in quale modo i messaggi inviati da un browser residente su un PC client vengono riportati sulla porta seriale del modulo ESP8266, utilizzando il monitor seriale disponibile nell'ambiente di Arduino.
Per impedire al micro di Arduino di interferire con la comunicazione diretta tra la finestra di monitor seriale e il modulo WiFi, al solito, si pongono in alta impedenza i suoi due pin di comunicazione caricando lo sketch che segue (**la compilazione degli sketch risulta più veloce se il canale WiFi tra il PC e il modulo risulta sconnesso**).

```
/* test ESP8266 */
void setup() {
pinMode (0, INPUT);
pinMode (1, INPUT);
}

voidloop() { }
```

Il monitor seriale va configurato per 115200 baud e con l'inserimento automatico dei due caratteri di ritorno carrello (CR) e di salta riga (NL) al termine di ogni comando inviato (**fig. 1**).

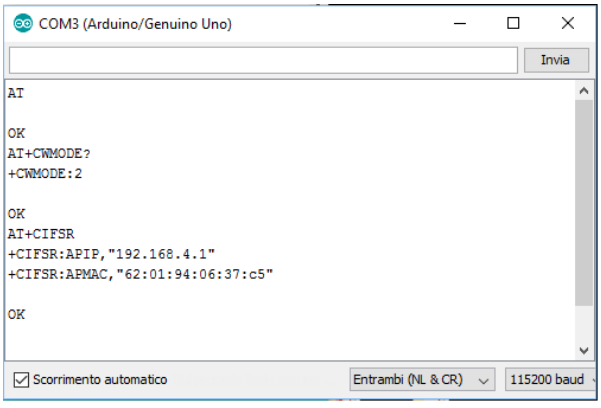

Fig. 1. *Predisposizione del monitor seriale.*

Connettere quindi il modulo ESP8266 con i pin 0 e 1 di Arduino, come già nell'esperienza precedente (**fig. 2**) e configurarlo come AP-Server sul port 80, con i comandi:

```
AT+CWMODE=2
AT+CIPMUX=1
AT+CIPSERVER=1,80
```

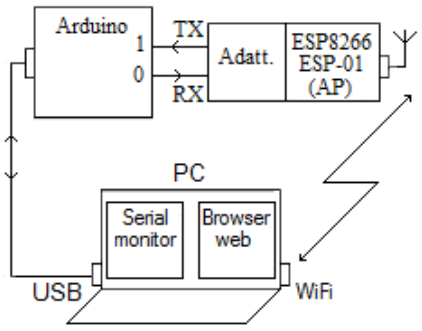

Fig. 2. *Modulo connesso al serial monitor.*

In modalità AP, il modulo ha un suo indirizzo IP di default (rilevabile con il comando AT+CIFSR) di valore 192.168.4.1.
Dopo aver richiesto e ottenuto dalla rete wireless del PC la connessione con l'access point Al-Thinker, aprendo, mediante browser web, una connessione http con l'indirizzo 192.168.4.1 (**fig. 3**), sul monitor di Arduino appaiono alcune interessanti informazioni (**fig. 4**).

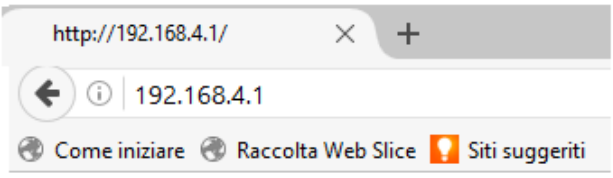

Fig. 3. *Apertura di un browser web.*

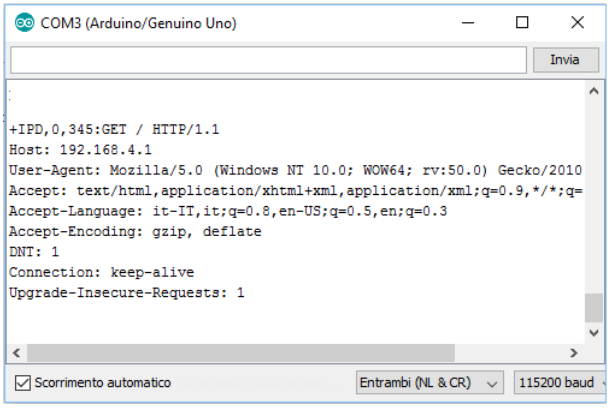

Fig. 4. *Informazioni relative al browser.*

Dopo i 5 caratteri "**+IPD,**", il numero della connessione ("**0**"), il numero dei caratteri trasferiti ("**345**") e il carattere "**:**" si ritrova il solito messaggio di apertura che inizia con i 5 caratteri "**GET /**" e altre informazioni, quali nome e versione del browser, il sistema operativo utilizzato, il tipo di file atteso, il linguaggio preferito, ecc.
Per inviare caratteri dal monitor seriale al browser del PC, digitare il solito comando:

```
AT+CIPSEND=0,5
```

e, all'apparire del carattere > (**fig. 5**), la stringa di 5 caratteri "ciaoo".
Sebbene il monitor avvisi che l'invio è andato a buon fine, sulla finestra del browser non appare nulla, finché (come richiesto dal protocollo) non si chiude il canale con il comando:

```
AT+CIPCLOSE=0
```

dove "**0**" indica appunto il numero del canale da chiudere.

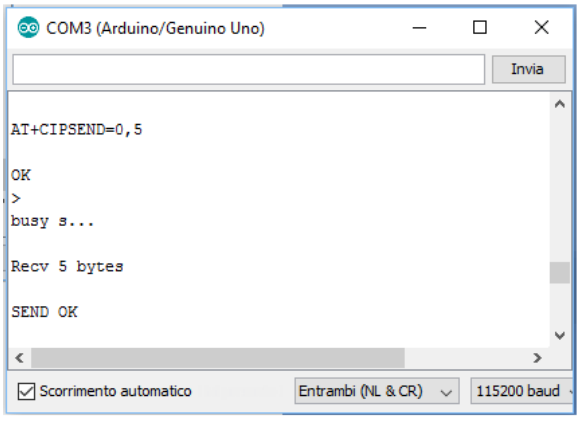

Fig. 5. *Invio al browser di 5 caratteri.*

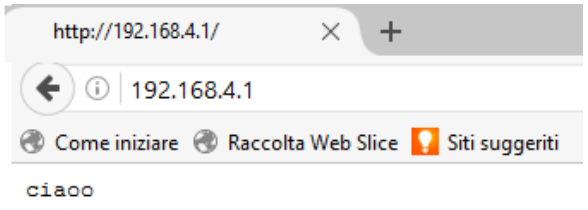

Fig. 6. *Messaggio visualizzato nella finestra del browser web.*

Invio comandi da pagina web

Per controllare Arduino mediante una pagina web con connessione WiFi, oltre a configurare il modulo ESP8266 in modalità operativa AP e avviare il servizio server sul port 80, serve un file HTML che, interpretato dal browser del PC, disegni sullo schermo una pagina grafica nella quale, per semplificare, premendo un bottone, si modifica lo stato di un'uscita di Arduino (**fig. 7**).

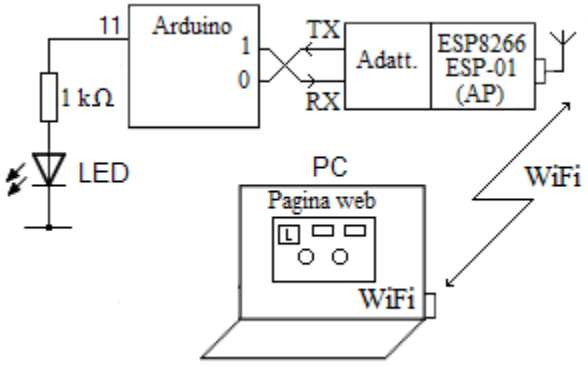

Fig. 7. *Comandi da pagina web.*

Questa volta, anziché il monitor seriale del PC, è il programma su Arduino che deve interagire in tempo reale con il modulo WiFi, pertanto **i segnali di comunicazione vanno incrociati**, come indicato in **fig. 8**, oltre a **sollevare il filo TX del modulo durante il caricamento del programma in Arduino**, per non confliggere con la TXD del PC.

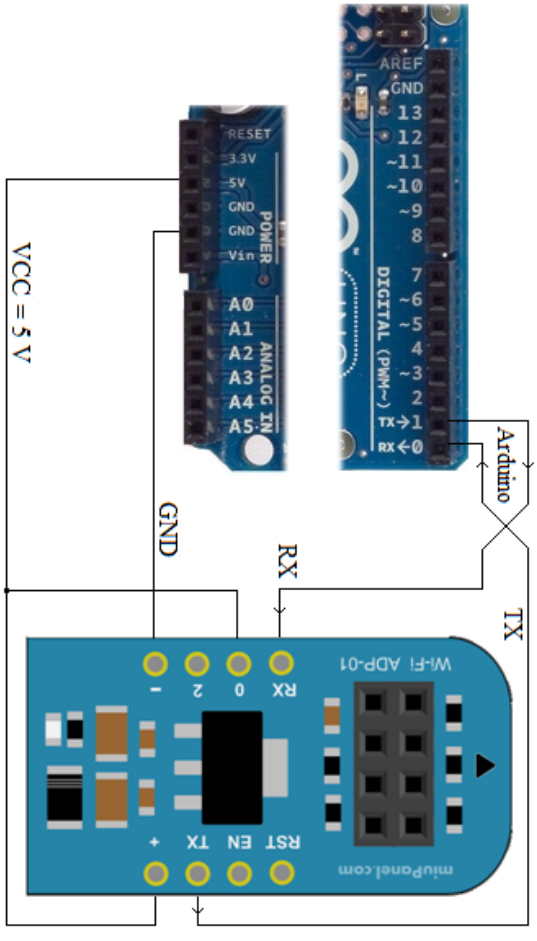

Fig. 8. *Connessioni seriali per funzionamento web server.*

Lo sketch per Arduino, riportato nel listato a lato, attende solo comandi contenenti la scritta "pin=11" per rovesciare lo stato logico di uscita del pin 11.

```
/* WiFi_html */
void setup()
{
digitalWrite(11,LOW);pinMode(11,OUTPUT);
Serial.begin(115200); // comunicazione con
ESP8266
Serial.print("AT+RST\r\n");
delay(1000); // resetta modulo
Serial.print("AT+CWMODE=2\r\n");
delay(500); // configura come access point
Serial.print("AT+CIPMUX=1\r\n");
delay(500); // configura per connessioni
multiple
Serial.print("AT+CIPSERVER=1,80\r\n");
delay(500); // avvia server sulla porta 80
}

voidloop()
{
if(Serial.available()) // controlla se il
modulo sta comunicando
 {
 if(Serial.find("+IPD,")) //Cerca nella
 stringa il testo "+IPD," inviato premendo
 il bottone nella pagina html
  {
  delay(20);
  int connectionId = Serial.read()-48; //al
  connection id va sottratto 48 ("0" ASCII)
  Serial.find("pin="); // adesso cerca "pin="
  int pinNumber = (Serial.read()-48)*10; //
  il primo numero sono le decine
  pinNumber += (Serial.read()-48); //il se-
  condo numero (unità) va sommato al primo
  digitalWrite(pinNumber,
  !digitalRead(pinNumber)); //rovescia lo
  stato del pin
  // compone il comando di chiusura da in-
  viare al modulo
  String closeCommand = "AT+CIPCLOSE=";
  closeCommand+=connectionId; // attacca il
  connection id
  closeCommand+="\r\n";
  Serial.print(closeCommand);
  delay(20);// attesa invio del comando di
  chiusura
  }
 }
}
```

Una volta caricato il programma (tenendo staccato il TX del modulo), ricollegare TX e aprire la finestra di monitor sul PC. Resettando la scheda, Arduino invia i comandi di configurazione al modulo WiFi (**fig. 9**) e resta in attesa.

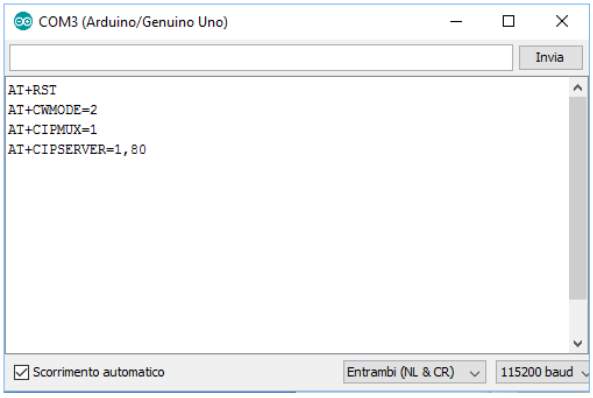

Fig. 9. *Comandi che Arduino invia al modulo WiFi.*

Il file HTML che, interpretato dal browser, comanda da PC (via WiFi) l'uscita 11 di Arduino è riportato di seguito.

```
<html>
<head>
<title>ESP8266 LED Control</title>
</head>
<body>
<!-- in the <button> tags below the ID at-
tribute is the value sent to the arduino -->
<button id="11" class="led">Toggle Pin 11</
button><!-- button for pin 11 -->

<script src="jquery.min.js"></script>
<script type="text/javascript">
$(document).ready(function(){
$(".led").click(function(){
var p = $(this).attr('id'); // get id value
(i.e. pin13, pin12, or pin11)
// invia la richiesta HTTP GET all'IP address
con il parametro "pin" di valore "p"
$.get("http://192.168.4.1:80/", {pin:p});
// execute get request
});
});
</script>

</body>
</html>
```

Occorre quindi:

- copiare il codice riportato in un documento mediante Notepad o qualsiasi altro text editor e salvarlo in formato testo in una cartella (per es. "*control.txt*");
- rinominare il file, modificandone l'estensione in ".html" (per es. "*control.html*");
- importare nella medesima cartella il file java *jquery.min* (**fig. 10**), copiandolo da http://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js

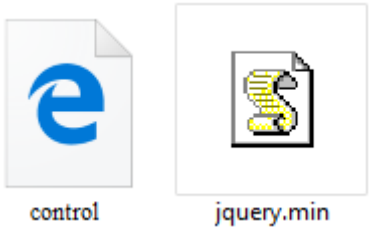

Fig. 10. *File .html e .min.*

Il codice HTML utilizzato usa difatti la libreria JavascriptJQuery, richiamata con l'istruzione:

```
<script src="jquery.min.js"></script>
```

A questo punto, dopo aver richiesto e ottenuto dalla rete wireless del PC la connessione con l'access point AI-Thinker, facendo doppio clic sul file *control.html*, compare la finestra di **fig. 11**.

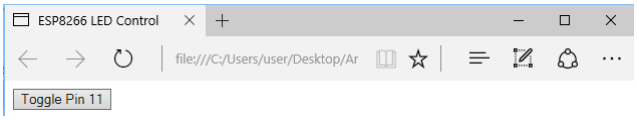

Fig. 11. *Pagina web visualizzata.*

Cliccando sul bottone "Toggle Pin 11", il programma invia una GET request al modulo ESP8266, mediante l'istruzione:

```
$.get("http://192.168.4.1:80/", {pin:p});
```

indicandone l'indirizzo IP e il port (80), con il numero 11 all'interno della variabile *p*.

Tramite la seriale, Arduino riceve una stringa di comunicazione del tipo:

```
+IPD,0,345:GET /?pin=11 HTTP/1.1
Host: 192.168.4.1
Connection: keep-alive
Accept:text/html,…
```

Al sopraggiungere di messaggi dalla linea seriale:

```
if(Serial.available())
```

Arduino identifica i comandi da elaborare controllando la presenza dello spezzone "+IPD," nel buffer ricevuto, mediante la funzione: `Serial.find`

```
if(Serial.find("+IPD,"))
```

Se ciò avviene, il codice legge il carattere successivo, corrispondente all'ID della connessione ("0" nell'esempio riportato), necessario per la corretta chiusura successiva della connessione stessa (richieste simultanee hanno ID differenti).

```
int connectionId = Serial.read()-48;
```

Successivamente il codice scorre la stringa alla ricerca dello spezzone "pin=" e acquisisce il numero del pin interessato, prima le decine e poi le unità:

```
Serial.find("pin=");
int pinNumber = (Serial.read()-48)*10;
pinNumber += (Serial.read()-48);
```

Al termine, rovescia il livello logico del pin indicato:

```
digitalWrite(pinNumber,
!digitalRead(pinNumber));
```

e chiude la connessione (**fig. 12**):

```
String closeCommand = "AT+CIPCLOSE=";
closeCommand+=connectionId; // attacca
il connection id
closeCommand+="\r\n";
Serial.print(closeCommand);
```

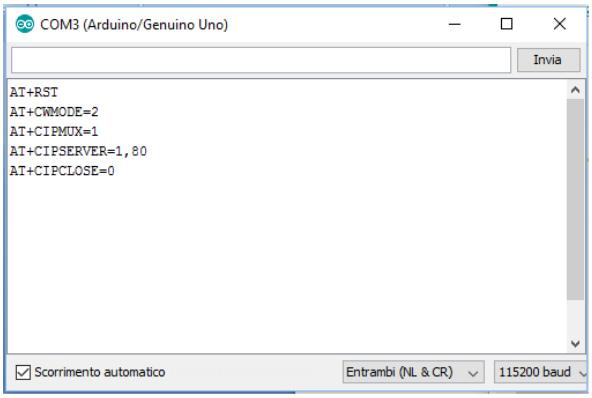

Fig. 12. *Invio del comando di chiusura.*

Possibile continuazione

Modificare lo sketch di Arduino e aggiungere altri due bottoni nella pagina HTML, in modo da controllare anche le uscite 12 e 13.

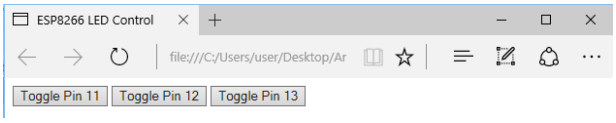

Fig. 13. *Nuova pagina visualizzata.*

Web server WiFi

Un dispositivo web server deve possedere una pagina HTML, da spedire all'indietro al client che chiede la connessione http, con all'interno le possibilità di interazione offerte al client.
Un server web è un dispositivo sempre in attesa di una richiesta da parte di un client, la serve e la chiude ogni volta.
Poiché in questo caso è interposta una connessione WiFi, Arduino deve gestire l'invio della pagina rispettando l'ordine previsto dal protocollo seriale del modulo ESP8266, ovvero deve prima inviare il comando:

```
AT+CIPSEND=c,n
```

con "c" l'identificativo del canale aperto e "n" il numero dei caratteri della pagina da trasferire, e quando appare il simbolo ">" (o con opportuno ritardo) inviare la pagina stessa.
Nello sketch che segue, l'invio di una pagina è delegato alla funzione **Send_webPage ()**, utilizzando il "**connectionId**" letto all'interno del messaggio ricevuto dal browser subito dopo la scritta "**+IPD,**".

```
/* Arduino WiFi server web */
int connectionId;
StringwebPage;
voidsetup()
{
Serial.begin(115200); // comunicazione con
ESP8266
Serial.print("AT+RST\r\n");
delay(1000); // resetta modulo
Serial.print("AT+CWMODE=2\r\n");
delay(500); // configura come access point
Serial.print("AT+CIPMUX=1\r\n");
delay(500); // configura per connessioni
multiple
Serial.print("AT+CIPSERVER=1,80\r\n");
delay(500); // avvia server sulla porta 80
}

voidloop()
{
if(Serial.available()) //controlla se il
modulo sta comunicando
 {
 if(Serial.find("+IPD,"))
  {
  delay(20);
  connectionId = Serial.read()-48;
  webPage = "<h1>Hello</h1><h2>World!</
  h2><button>LED1</button>";
  Send_webPage ();
  webPage="<button>LED2</button>";
  Send_webPage ();
  Send_closeCommand();
  }
 }
}

void Send_webPage(){
String sendCommand = "AT+CIPSEND=";
sendCommand += connectionId;
sendCommand += ",";
sendCommand +=webPage.length();
sendCommand +="\r\n";
Serial.print(sendCommand);
delay(500);
Serial.print(webPage);
delay(500);
}
void Send_closeCommand(){
String closeCommand = "AT+CIPCLOSE=";
closeCommand+=connectionId;
closeCommand+="\r\n";
Serial.print(closeCommand);
delay(500);
}
```

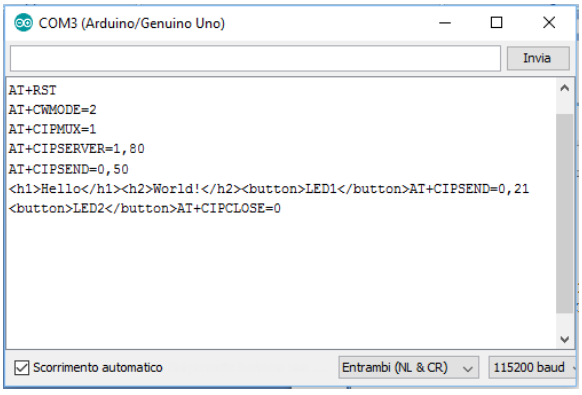

Fig. 14. *Comandi inviati da Arduino.*

Una volta caricato il programma (tenendo staccato il TX del modulo), ricollegare TX e aprire la finestra di monitor sul PC. Resettando la scheda, Arduino invia i comandi di configurazione al modulo WiFi e resta in attesa.
A questo punto, attivare la connessione WiFi tra il PC e Al-Thinker e aprire un browser all'indirizzo 192.168.4.1.

Sul monitor seriale compare la sequenza di invio della pagina di risposta (**fig. 14**) e sul browser la sua interpretazione grafica (**fig. 15**).

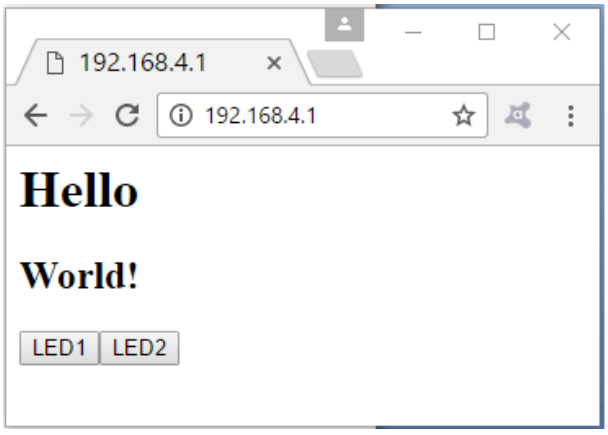

Fig. 15. *Pagina visualizzata dal browser.*

Pagina web WiFi interattiva

Per ottenere una connessione che permetta il controllo del processo su Arduino, i bottoni vanno resi capaci di scatenare una richiesta Get e il browser del client (PC) va istruito in modo da inviare una richiesta http con periodicità fissa e in modo automatico, per l'aggiornamento in tempo reale dello stato degli ingressi residenti sul server (Arduino), includendo nella pagina HTML l'istruzione **Refresh**.

Il programma che segue, per esempio, gestisce le richieste provenienti da due bottoni per il governo di un LED posto sul pin 4 di Arduino (**fig. 16**), e aggiorna ogni 5 secondi (Refresh: 5) il client sullo stato dell'ingresso digitale 2 e dell'ingresso analogico A0.

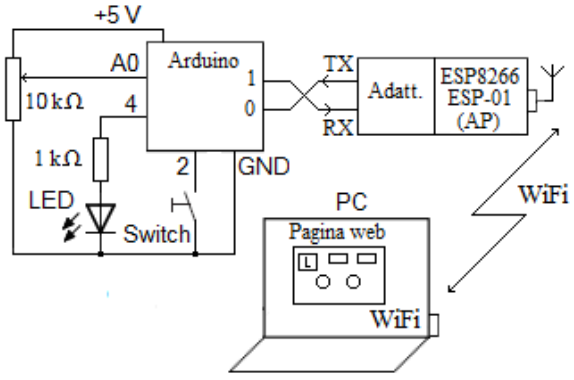

Fig. 16. *Web server WiFi.*

```
/* Arduino WiFi server web_2 */
int connectionId, i;
String webPage;
char str[100];
char* StatoLed4;

voidsetup()
{
Serial.begin(115200); // comunicazione con
ESP8266
Serial.print("AT+RST\r\n");
delay(1000); // resetta modulo
Serial.print("AT+CWMODE=2\r\n");
delay(500); // configura come access point
Serial.print("AT+CIPMUX=1\r\n");
delay(500); // configura per connessioni
multiple
Serial.print("AT+CIPSERVER=1,80\r\n");
delay(500); // avvia server sulla porta 80
pinMode(2, INPUT_PULLUP); //pin 2 input con
pull-up
  //digitalWrite(2, HIGH);
pinMode(4, OUTPUT);
digitalWrite(4, LOW);
}

voidloop()
{
if(Serial.available()) //controlla se il
modulo sta comunicando
 {
 if(Serial.find("+IPD,"))
  {
  delay(20);
  connectionId = Serial.read()-48;
   if(Serial.find("GET /?")) {
    for(i=0;i<7;i++)str[i]=Serial.read();
    str[i]=0;
     if(strstr(str,"pin4=on"))digi-
     talWrite(4, HIGH);
     else if(strstr(str,"pin4=of"))digi-
     talWrite(4, LOW);
    }
   if(digitalRead(4))StatoLed4="on";
   else StatoLed4="off";
   homePage();// prepara la pagina
   Send_webPage ();
   Send_closeCommand();
   }
  }
}
```

```
void Send_webPage(){
String sendCommand = "AT+CIPSEND=";
sendCommand += connectionId;
sendCommand += ",";
sendCommand +=webPage.length();
sendCommand +="\r\n";
Serial.print(sendCommand);
  while (Serial.read ()!= '>');
Serial.print(webPage);
delay(200);
}

void Send_closeCommand(){
String closeCommand = "AT+CIPCLOSE=";
closeCommand+=connectionId;
closeCommand+="\r\n";
Serial.print(closeCommand);
delay(300);
}

static word homePage() {
long t = millis() / 1000;
word h = t / 3600;
byte m = (t / 60) % 60;
byte s = t % 60;
webPage="HTTP/1.0 200 OK\r\n"
"Content-Type: text/html\r\nPragma: no-
cache\r\nRefresh: 5\r\n\r\n"
"<html><head><title>ENC28J60</title></head>"
"<body>"
"<div style='text-align:center;'>"
"<h1>ENC28J60 web server</h1>"
"<h4>Durata della connessione :";
sprintf(str,"%d%d:%d%d:%d%d</h4>",h/10,
h%10, m/10, m%10, s/10, s%10);
webPage.concat(str);
sprintf(str,"<br />Input 2 = %d<br />",digi-
talRead(2));
webPage.concat(str);
sprintf(str,"<br /><br />Stato del LED4:
%s<br />",StatoLed4);
webPage.concat(str);
webPage+="<a  href='/?pin4=on'><input
type='button' value='OUT 4 ON'></a>"
"<a href='/?pin4=off'><input type='button'
value='OUT 4 OFF'></a>"
"<br /><br />";
sprintf(str,"<br /><br />Potenziometro: %d
(su 1024)",analogRead(0));
webPage.concat(str);
webPage+="<br /><br /></body></html>";
}
```

Una volta caricato lo sketch, aprendo la connessione WiFi e attivando il browser web all'indirizzo 192.168.4.1, comparirà la schermata riportata in **fig. 17**, con il tempo di connessione, lo stato logico dell'ingresso digitale, la possibilità di modificare lo stato del LED e il livello rilevato su A0.

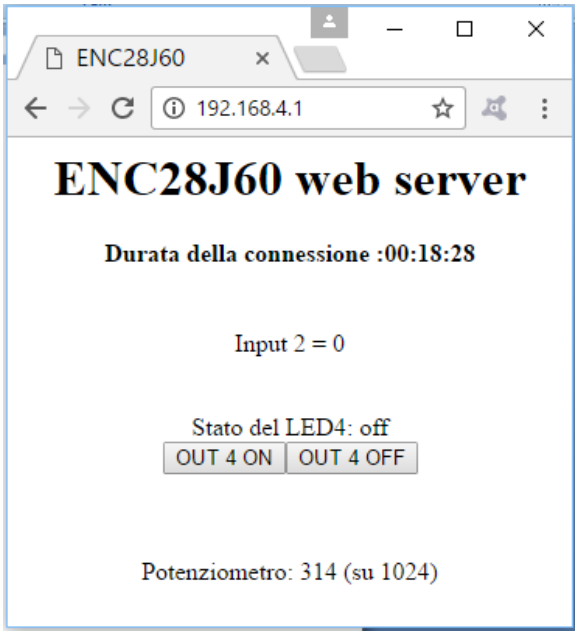

Fig. 17. *Pagina visualizzata.*

Ogni 5 secondi o premendo uno dei pulsanti, il PC attiva una richiesta di connessione e invia un comando. Se nella stringa inviata dal client (**fig. 18**) lo sketch rileva lo spezzone "+IPD,", salva il numero della connessione, gestisce l'eventuale richiesta GET/? (se presente), invia una pagina HTML con i valori aggiornati e chiude la comunicazione.

```
+IPD,1,437:GET /?pin4=off HTTP/1.1
Host: 192.168.4.1 …
```

Fig. 18. *Esempio parziale di stringa ricevuta da Arduino.*

Possibile continuazione

Aggiungere un ulteriore switch attivo basso sul pin 3, un ulteriore LED sul pin 5 e modificare il programma in modo da renderli entrambi interattivi come già fatto per i precedenti.