1. Protocollo HTTP: Concetti di Base

1.1 Cos'è il protocollo HTTP

HTTP (HyperText Transfer Protocol) è un protocollo a livello applicativo utilizzato per la trasmissione di dati sul web. È basato su un modello client-server in cui:

- Il **client** (browser) invia richieste al server
- Il **server** elabora le richieste e risponde inviando risorse

1.2 Caratteristiche principali

- **Stateless**: ogni richiesta è indipendente dalle precedenti
- **Basato su testo**: le richieste e risposte sono in formato testuale
- **Client-server**: separazione netta dei ruoli
- **Connectionless**: dopo la risposta, la connessione viene chiusa (HTTP/1.0) o può essere mantenuta (HTTP/1.1 con keep-alive)

1.3 Composizione di un pacchetto HTTP

Richiesta HTTP

```
METODO /risorsa HTTP/versione
Header1: valore1
Header2: valore2
...

corpo del messaggio (opzionale)
```

Esempio:

```
GET /index.html HTTP/1.1
Host: www.esempio.it
User-Agent: Mozilla/5.0
Accept: text/html
```

Risposta HTTP

```
HTTP/versione CODICE DESCRIZIONE
Header1: valore1
Header2: valore2
```

```
...

corpo del messaggio (contenuto richiesto)
```

Esempio:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1234

<!DOCTYPE html>
<html>
...
</html>
```

1.4 Metodi HTTP

- **GET**: richiede una risorsa
- **POST**: invia dati al server (es. form)
- **PUT**: aggiorna una risorsa sul server
- **DELETE**: elimina una risorsa
- **HEAD**: simile a GET ma richiede solo gli header, senza body

1.5 Codici di stato

- **1xx**: Informational (richiesta ricevuta, elaborazione in corso)
- **2xx**: Success (200 OK, 201 Created)
- **3xx**: Redirection (301 Moved Permanently, 302 Found)
- **4xx**: Client Error (404 Not Found, 403 Forbidden)
- **5xx**: Server Error (500 Internal Server Error)

2. ESP32 come WebServer

2.1 Modalità di funzionamento dell'ESP32

L'ESP32 può operare come:

- **Station (STA)**: client in una rete esistente
- **Access Point (AP)**: crea una propria rete
- **STA+AP**: entrambe le modalità contemporaneamente

2.2 Configurazione base dell'ESP32

Station Mode (Client)

```
#include <WiFi.h>

const char* ssid = "NomeRete";
const char* password = "Password";

void setup() {
  Serial.begin(115200);

  // Connessione WiFi
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connesso");
  Serial.println("Indirizzo IP: " + WiFi.localIP().toString());
}
```

Access Point Mode

```
#include <WiFi.h>

const char* ssid = "ESP32_AP";
const char* password = "password123";

void setup() {
  Serial.begin(115200);

  // Configurazione Access Point
  WiFi.softAP(ssid, password);

  Serial.println("Access Point avviato");
  Serial.print("SSID: ");
  Serial.println(ssid);
  Serial.print("Indirizzo IP: ");
  Serial.println(WiFi.softAPIP().toString());
}
```

2.3 DHCP vs IP Statico

IP Dinamico (DHCP)

```
WiFi.begin(ssid, password);
// L'indirizzo IP viene assegnato automaticamente dal DHCP
Serial.println(WiFi.localIP());
```

IP Statico

```
IPAddress ip(192, 168, 1, 100);     // IP desiderato
IPAddress gateway(192, 168, 1, 1);   // Gateway (router)
IPAddress subnet(255, 255, 255, 0);  // Subnet mask
IPAddress dns(8, 8, 8, 8);           // DNS (Google)

// Configurazione IP statico prima della connessione
WiFi.config(ip, gateway, subnet, dns);
WiFi.begin(ssid, password);
```

3. Implementazione di un WebServer base

3.1 Utilizzo della libreria WiFiServer

```
#include <WiFi.h>

const char* ssid = "NomeRete";
const char* password = "Password";

// Server sulla porta HTTP standard
WiFiServer server(80);

void setup() {
  Serial.begin(115200);

  // Connessione WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connesso");
  Serial.println("Indirizzo IP: " + WiFi.localIP().toString());

  // Avvio del server
  server.begin();
}

void loop() {
  // Verifica se un client si è connesso
```

```
  WiFiClient client = server.available();

  if (client) {
    Serial.println("Nuovo client connesso");

    // Lettura della richiesta HTTP
    String currentLine = "";
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        Serial.write(c);

        // Se riceviamo un carattere di nuova riga
        if (c == '\n') {
          // Se la linea corrente è vuota, significa fine degli header
          if (currentLine.length() == 0) {
            // Invio degli header HTTP
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();

            // Invio del contenuto della pagina
            client.println("<!DOCTYPE html><html>");
            client.println("<head><meta name=\"viewport\"
content=\"width=device-width, initial-scale=1\">");
            client.println("<title>ESP32 Web Server</title>");
            client.println("</head><body>");
            client.println("<h1>ESP32 Web Server</h1>");
            client.println("<p>Esempio di pagina web servita da ESP32</p>");
            client.println("</body></html>");

            // La risposta si conclude qui
            break;
          } else {
            // Nuova linea, svuotiamo la precedente
            currentLine = "";
          }
        } else if (c != '\r') {
          // Aggiungi il carattere alla linea corrente
          currentLine += c;
        }
      }
    }

    // Chiusura della connessione
    client.stop();
    Serial.println("Client disconnesso");
  }
}
```

3.2 Utilizzo della libreria WebServer

La libreria WebServer è più avanzata e semplifica la gestione delle richieste HTTP:

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

const char* ssid = "NomeRete";
const char* password = "Password";

// Creazione dell'oggetto server sulla porta 80
WebServer server(80);

void setup() {
  Serial.begin(115200);

  // Connessione alla rete Wi-Fi
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connesso");
  Serial.println("Indirizzo IP: " + WiFi.localIP().toString());

  // Definizione delle route
  server.on("/", handleRoot);
  server.on("/led/on", handleLedOn);
  server.on("/led/off", handleLedOff);
  server.onNotFound(handleNotFound);

  // Avvio del server
  server.begin();
  Serial.println("Server HTTP avviato");
}

void loop() {
  // Gestione delle richieste client
  server.handleClient();
}

// Gestore della pagina principale
void handleRoot() {
  String html = "<!DOCTYPE html><html>";
  html += "<head><meta name=\"viewport\" content=\"width=device-width,
```

```
initial-scale=1\">";
  html += "<title>ESP32 Web Server</title></head>";
  html += "<body><h1>ESP32 Web Control</h1>";
  html += "<p><a href=\"/led/on\"><button>LED ON</button></a></p>";
  html += "<p><a href=\"/led/off\"><button>LED OFF</button></a></p>";
  html += "</body></html>";

  server.send(200, "text/html", html);
}

// Gestore per accendere il LED
void handleLedOn() {
  // Codice per accendere il LED
  digitalWrite(LED_BUILTIN, HIGH);

  server.send(200, "text/plain", "LED acceso");
}

// Gestore per spegnere il LED
void handleLedOff() {
  // Codice per spegnere il LED
  digitalWrite(LED_BUILTIN, LOW);

  server.send(200, "text/plain", "LED spento");
}

// Gestore per pagine non trovate (404)
void handleNotFound() {
  server.send(404, "text/plain", "Pagina non trovata");
}
```

4. Gestione di richieste GET e POST

4.1 Parametri GET

I parametri GET vengono inviati attraverso l'URL:

```
http://192.168.1.100/sensore?id=1&valore=25
```

Codice per leggere i parametri GET:

```
server.on("/sensore", HTTP_GET, []() {
  String id = server.arg("id");
  String valore = server.arg("valore");

  Serial.println("ID: " + id);
  Serial.println("Valore: " + valore);
```

```
  server.send(200, "text/plain", "Parametri ricevuti: ID=" + id + ",
Valore=" + valore);
});
```

4.2 Parametri POST

I parametri POST vengono inviati nel corpo della richiesta:

```
server.on("/submit", HTTP_POST, []() {
  String username = server.arg("username");
  String password = server.arg("password");

  Serial.println("Username: " + username);
  Serial.println("Password: " + password);

  server.send(200, "text/plain", "Dati ricevuti");
});
```

HTML form di esempio:

```
<form action="/submit" method="post">
  <input type="text" name="username">
  <input type="password" name="password">
  <input type="submit" value="Invia">
</form>
```

4.3 Lettura del corpo di una richiesta POST

Per leggere il corpo di una richiesta POST in formato raw:

```
server.on("/api/data", HTTP_POST, []() {
  String payload = server.arg("plain");
  Serial.println("Dati ricevuti: " + payload);

  // Elaborazione dei dati
  server.send(200, "text/plain", "Dati ricevuti correttamente");
});
```

5. Pagina di Login e Autenticazione

5.1 Creazione di una pagina di login

```
void handleRoot() {
  String html = "<!DOCTYPE html><html>";
```

```
  html += "<head><meta name=\"viewport\" content=\"width=device-width,
initial-scale=1\">";
  html += "<title>Login</title></head>";
  html += "<body><h1>Login</h1>";
  html += "<form action=\"/login\" method=\"post\">";
  html += "Username: <input type=\"text\" name=\"username\"><br>";
  html += "Password: <input type=\"password\" name=\"password\"><br>";
  html += "<input type=\"submit\" value=\"Login\">";
  html += "</form></body></html>";

  server.send(200, "text/html", html);
}
```

5.2 Gestione dell'autenticazione

```
// Credenziali valide
const char* valid_username = "admin";
const char* valid_password = "password";

// Flag per lo stato del login
bool authenticated = false;

void handleLogin() {
  if (server.method() != HTTP_POST) {
    server.send(405, "text/plain", "Method Not Allowed");
    return;
  }

  String username = server.arg("username");
  String password = server.arg("password");

  if (username == valid_username && password == valid_password) {
    authenticated = true;
    server.sendHeader("Location", "/dashboard");
    server.send(303);
  } else {
    server.send(200, "text/html", "<html><body><p>Login fallito</p><a
href='/'>Riprova</a></body></html>");
  }
}

void handleDashboard() {
  if (!authenticated) {
    server.sendHeader("Location", "/");
    server.send(303);
    return;
  }
```

```
  String html = "<!DOCTYPE html><html>";
  html += "<head><title>Dashboard</title></head>";
  html += "<body><h1>Dashboard</h1>";
  html += "<p>Benvenuto nella dashboard protetta!</p>";
  html += "<a href='/logout'>Logout</a>";
  html += "</body></html>";

  server.send(200, "text/html", html);
}

void handleLogout() {
  authenticated = false;
  server.sendHeader("Location", "/");
  server.send(303);
}
```

5.3 Autenticazione HTTP Basic

Un metodo alternativo utilizzando l'autenticazione HTTP Basic:

```
server.on("/area-protetta", []() {
  if (!server.authenticate("admin", "password")) {
    return server.requestAuthentication();
  }
  server.send(200, "text/plain", "Area protetta - Accesso autorizzato");
});
```

6. Creazione di una Dashboard per monitoraggio

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

const char* ssid = "NomeRete";
const char* password = "Password";

WebServer server(80);

// Simulazione sensore
float getTemperature() {
  return random(10, 35) + random(10) / 10.0; // Simula temperatura tra 10°C
e 35°C
}

void setup() {
  Serial.begin(115200);

  // Connessione WiFi
```

```
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connesso");
  Serial.println("Indirizzo IP: " + WiFi.localIP().toString());

  // Route per pagina principale
  server.on("/", handleRoot);

  // API per ottenere temperatura attuale
  server.on("/api/temperatura", handleTemperatura);

  server.begin();
}

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

void handleRoot() {
  String html = "<!DOCTYPE html><html>";
  html += "<head>";
  html += "<meta name='viewport' content='width=device-width, initial-
scale=1'>";
  html += "<title>Dashboard Temperatura</title>";
  html += "<style>";
  html += "body { font-family: Arial; text-align: center; margin-top: 50px;
}";
  html += "#temp { font-size: 48px; font-weight: bold; }";
  html += "</style>";
  html += "<script>";
  html += "function aggiornaDati() {";
  html += "  fetch('/api/temperatura')";
  html += "    .then(response => response.text())";
  html += "    .then(data => {";
  html += "      document.getElementById('temp').innerHTML = data + '°C';";
  html += "    });";
  html += "}";
  html += "setInterval(aggiornaDati, 5000);"; // Aggiorna ogni 5 secondi
  html += "document.addEventListener('DOMContentLoaded', aggiornaDati);";
  html += "</script>";
  html += "</head>";
  html += "<body>";
  html += "<h1>Dashboard Temperatura</h1>";
  html += "<div id='temp'>--.-°C</div>";
  html += "<p>Aggiornamento automatico ogni 5 secondi</p>";
```

```
  html += "</body></html>";

  server.send(200, "text/html", html);
}

void handleTemperatura() {
  float temp = getTemperature();
  server.send(200, "text/plain", String(temp, 1));
}
```

7. Risposte HTTP avanzate

7.1 Reindirizzamenti (Redirect)

```
server.sendHeader("Location", "/nuova-pagina");
server.send(302, "text/plain", ""); // 302 Found
```

7.2 Risposte con cookie

```
server.sendHeader("Set-Cookie", "sessione=12345");
server.send(200, "text/html", "<h1>Cookie impostato</h1>");
```

7.3 Risposte con diversi tipi di contenuto

```
// Risposta HTML
server.send(200, "text/html", "<h1>Hello World!</h1>");

// Risposta JSON
server.send(200, "application/json", "{\"status\":\"ok\",\"valore\":42}");

// Risposta XML
server.send(200, "text/xml", "<?xml version=\"1.0\"?><dati>
<temperatura>25</temperatura></dati>");
```

8. Best Practices per WebServer su ESP32

1. **Ottimizzazione della memoria**
   - Usare SPIFFS o SD per file statici grandi
   - Minimizzare le stringhe HTML
   - Usare F() per stringhe costanti (es: `server.send(200, F("text/html"), F("<h1>Titolo</h1>"));`)
2. **Gestione delle connessioni**
   - Impostare timeout per le connessioni client
   - Evitare blocchi lunghi nel loop principale

3. **Sicurezza**
    - Implementare autenticazione per funzioni critiche
    - Usare HTTPS se necessario (richiede più risorse)
    - Verificare input utente
4. **Prestazioni**
    - Utilizzare cache-control per risorse statiche
    - Minimizzare dimensioni pagine HTML e JavaScript
    - Considerare l'uso di webserver asincroni per applicazioni complesse

9. Conclusione

L'ESP32 è un potente microcontrollore che permette di implementare webserver HTTP completi per applicazioni IoT. Attraverso il protocollo HTTP, è possibile creare interfacce web interattive per monitorare e controllare dispositivi connessi.

Le librerie WiFi e WebServer di Arduino semplificano notevolmente questo processo, permettendo di concentrarsi sulla logica applicativa piuttosto che sui dettagli del protocollo di rete.

Riferimenti

- Documentazione ufficiale ESP32
- Libreria WebServer per ESP32
- Specifiche HTTP/1.1 (RFC 2616)