Fondamenti teorici

Architettura client-server HTTP

- **Client**: dispositivo che richiede risorse o servizi (browser, applicazione o microcontrollore)
- **Server**: sistema che risponde alle richieste fornendo risorse o servizi
- **HTTP**: protocollo applicativo che definisce il formato delle comunicazioni

ESP32 e capacità di rete

- **ESP32**: microcontrollore SoC con Wi-Fi e Bluetooth integrati
- Può fungere sia da client HTTP (richiedere dati) che da server HTTP (fornire servizi)
- Utilizza le librerie `WiFi.h` e `WebServer.h` per le funzionalità di rete

Tipi di richieste HTTP

1. **GET**: richiesta di risorse, parametri inviati nell'URL
2. **POST**: invio di dati al server (form, autenticazioni), parametri nel corpo
3. **PUT**: aggiornamento completo di risorse
4. **DELETE**: eliminazione di risorse
5. **PATCH**: aggiornamento parziale di risorse

Codici di risposta HTTP

- **2xx**: successo (200 OK, 201 Created)
- **3xx**: redirezione
- **4xx**: errore client (404 Not Found)
- **5xx**: errore server

Implementazioni ESP32

ESP32 come client HTTP

Un client HTTP si connette a un server per ottenere risorse o inviare dati. Il flusso operativo è:

1. Connessione alla rete Wi-Fi
2. Apertura connessione TCP verso server
3. Invio richiesta HTTP formattata
4. Ricezione e gestione della risposta

ESP32 come server HTTP

Un server HTTP attende e gestisce richieste in ingresso. Il flusso operativo è:

1. Connessione alla rete Wi-Fi
2. Configurazione gestori (handlers) per vari endpoint
3. Avvio del server in ascolto
4. Gestione continua delle richieste client in arrivo

Gestione form e autenticazione

La gestione di form e autenticazione richiede:

1. Definizione di pagine HTML con elementi form
2. Ricezione e parsing dei parametri (via GET o POST)
3. Verifica delle credenziali e gestione delle sessioni
4. Reindirizzamento alle pagine appropriate

Struttura delle richieste HTTP

Formato generale

```
METODO /risorsa HTTP/versione
Header1: valore1
Header2: valore2
...
<linea vuota>
corpo della richiesta (opzionale)
```

Esempio richiesta GET

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: ESP32/1.0
Connection: close
```

Esempio richiesta POST

```
POST /login HTTP/1.1
Host: www.example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 32
```

```
username=admin&password=secret123
```

Struttura delle risposte HTTP

Formato generale

```
HTTP/versione codice_stato messaggio
Header1: valore1
Header2: valore2
...
<linea vuota>
corpo della risposta (opzionale)
```

Esempio risposta

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 2048
Connection: close

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

Gestione delle richieste in ESP32

Ciclo di vita della richiesta

1. **Ricezione**: Il server ESP32 riceve la richiesta HTTP tramite socket
2. **Parsing**: Analisi dell'intestazione e separazione dei parametri
3. **Routing**: Determinazione della funzione handler appropriata
4. **Gestione**: Esecuzione della funzione handler associata all'URI
5. **Risposta**: Generazione e invio della risposta al client

Parsing dei parametri

ESP32 WebServer supporta sia parametri GET (nell'URL) che POST (nel corpo):

- **Parametri GET**: `http://esempio.com/pagina?param1=valore1&param2=valore2`
- **Parametri POST**: Inviati nel corpo della richiesta con un form HTML

Accesso ai parametri con ESP32

```
// Numero totale dei parametri
int numParams = server.args();

// Lettura di parametri specifici
String username = server.arg("username");
String password = server.arg("password");

// Iterazione su tutti i parametri
for (int i = 0; i < server.args(); i++) {
  String paramName = server.argName(i);
  String paramValue = server.arg(i);
}
```

Tipi MIME nelle risposte

I tipi MIME (Multipurpose Internet Mail Extensions) indicano il formato del contenuto:

Tipo MIME	Descrizione
text/html	Documento HTML
text/plain	Testo semplice
application/json	Dati in formato JSON
image/jpeg	Immagine JPEG
application/pdf	Documento PDF

Utilizzo in ESP32:

```
server.send(200, "text/html", "<html><body>Contenuto</body></html>");
server.send(200, "application/json", "{\"stato\":\"ok\"}");
```

Tecniche di autenticazione

Basic Authentication

- Credenziali inviate in ogni richiesta (username:password in base64)
- Header: `Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=`
- Sicurezza limitata senza HTTPS

Form-based Authentication

- Credenziali inviate tramite form HTML

- Verifica lato server e creazione di sessione
- Implementazione in ESP32 tramite controllo parametri POST

Token-based Authentication

- Token generato dopo login e inviato in ogni richiesta
- Richiede gestione dello stato sul server
- Implementazione avanzata con ESP32 richiede gestione manuale

Sessioni e stato

HTTP è un protocollo stateless, ma è possibile implementare sessioni:

1. **Cookie**: Inviati dal server, memorizzati e restituiti dal client
2. **Parametri URL**: Inclusione di identificatori di sessione nell'URL
3. **Variabili di stato**: Memorizzazione lato server associata a identificatori

Su ESP32, l'implementazione di sessioni richiede:

- Generazione di ID sessione univoci
- Memorizzazione dello stato in strutture dati
- Associazione tra client e dati di sessione

WebServer completo

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

// Configurazione rete Wi-Fi
const char *ssid = "your_ssid";
const char *password = "your_password";

// Istanza del server web sulla porta 80 (HTTP)
WebServer server(80);

// === DEFINIZIONE PAGINE HTML ===
// Pagina di login
String loginPage =
"<!DOCTYPE html>"
"<html>"
"   <head><title>ESP32 Login</title></head>"
"   <body>"
"     <h2>ESP32 Login</h2>"
"     <form action='/login' method='post'>"
"       <label>Username:</label><input type='text' name='username'><br>"
"       <label>Password:</label><input type='password' name='password'><br>
```

```
<br>"
"       <input type='submit' value='Accedi'>"
"     </form>"
"   </body>"
"</html>";

// Pagina di errore login
String loginPageError =
"<!DOCTYPE html>"
"<html>"
"   <head><title>ESP32 Login Error</title></head>"
"   <body>"
"     <h2>ESP32 Login</h2>"
"     <p style='color:red;'>Username o password errati</p>"
"     <form action='/login' method='post'>"
"       <label>Username:</label><input type='text' name='username'><br>"
"       <label>Password:</label><input type='password' name='password'><br>
<br>"
"       <input type='submit' value='Accedi'>"
"     </form>"
"   </body>"
"</html>";

// Pagina admin
String adminPage =
"<!DOCTYPE html>"
"<html>"
"   <head><title>Admin Panel</title></head>"
"   <body>"
"     <h2>Pannello Amministratore</h2>"
"     <p>Benvenuto, Admin!</p>"
"     <div>"
"       <h3>Controllo Dispositivi</h3>"
"       <a href='/toggle/led'>Toggle LED</a>"
"     </div>"
"     <a href='/logout'>Logout</a>"
"   </body>"
"</html>";

// Pagina utente
String userPage =
"<!DOCTYPE html>"
"<html>"
"   <head><title>User Panel</title></head>"
"   <body>"
"     <h2>Pannello Utente</h2>"
"     <p>Benvenuto, Utente!</p>"
"     <div>"
"       <h3>Stato Dispositivo</h3>"
"       <p id='status'>Stato: normale</p>"
```

```
"     </div>"
"     <a href='/logout'>Logout</a>"
"   </body>"
"</html>";

// === HANDLERS DELLE RICHIESTE HTTP ===

// Handler per la pagina principale
void handleRoot() {
  server.send(200, "text/html", loginPage);
}

// Handler per la pagina di login
void handleLogin() {
  bool isAdmin = false;
  bool isUser = false;

  // Verifica le credenziali inviate tramite POST
  if (server.method() == HTTP_POST) {
    for (uint8_t i = 0; i < server.args(); i++) {
      // Verifica credenziali amministratore
      if (server.argName(i) == "username" && server.arg(i) == "admin" &&
          server.hasArg("password") && server.arg("password") == "admin123")
{
        isAdmin = true;
      }
      // Verifica credenziali utente normale
      if (server.argName(i) == "username" && server.arg(i) == "user" &&
          server.hasArg("password") && server.arg("password") == "user123")
{
        isUser = true;
      }
    }

    // Reindirizza in base al tipo di utente
    if (isAdmin) {
      server.send(200, "text/html", adminPage);
    } else if (isUser) {
      server.send(200, "text/html", userPage);
    } else {
      // Credenziali non valide
      server.send(200, "text/html", loginPageError);
    }
  } else {
    // Se non è una richiesta POST, mostra la pagina di login
    server.send(200, "text/html", loginPage);
  }
}

// Handler per le richieste non trovate (404)
```

```
void handleNotFound() {
  String message = "Pagina Non Trovata\n\n";
  message += "URI: ";
  message += server.uri();
  message += "\nMetodo: ";
  message += (server.method() == HTTP_GET) ? "GET" : "POST";
  message += "\nParametri: ";
  message += server.args();
  message += "\n";

  for (uint8_t i = 0; i < server.args(); i++) {
    message += " " + server.argName(i) + ": " + server.arg(i) + "\n";
  }

  server.send(404, "text/plain", message);
}

// Handler per una richiesta GET con parametri
void handleParams() {
  String message = "Parametri ricevuti:\n";

  // Itera su tutti i parametri ricevuti
  for (uint8_t i = 0; i < server.args(); i++) {
    message += " - " + server.argName(i) + ": " + server.arg(i) + "\n";
  }

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

// Handler per richieste API in formato JSON
void handleApi() {
  // Verifica il metodo della richiesta
  if (server.method() == HTTP_GET) {
    // Esempio di risposta JSON per GET
    server.send(200, "application/json", "{\"status\":\"success\",\"data\":
{\"temp\":22.5,\"humidity\":45}}");
  }
  else if (server.method() == HTTP_POST) {
    // Lettura dei dati JSON inviati dal client
    String jsonData = "";

    if (server.hasArg("plain")) {
      jsonData = server.arg("plain");

      // Qui si processerebbe il JSON (in un'implementazione reale)
      // Per semplicità, risponde con l'eco dei dati ricevuti
      server.send(200, "application/json", "
{\"status\":\"received\",\"data\":" + jsonData + "}");
    } else {
      server.send(400, "application/json", "
```

```
{\"status\":\"error\",\"message\":\"No data\"}");
    }
  }
  else {
    // Metodo non supportato
    server.send(405, "application/json", "
{\"status\":\"error\",\"message\":\"Method not allowed\"}");
  }
}

// === SETUP E LOOP PRINCIPALI ===

void setup() {
  // Inizializza la comunicazione seriale
  Serial.begin(115200);

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

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

  Serial.println();
  Serial.print("Connesso! Indirizzo IP: ");
  Serial.println(WiFi.localIP());

  // Configura mDNS per accesso facilitato
  if (MDNS.begin("esp32")) {
    Serial.println("Servizio mDNS avviato - accesso tramite
http://esp32.local");
  }

  // Configura gli handler delle richieste
  server.on("/", HTTP_GET, handleRoot);
  server.on("/login", handleLogin);
  server.on("/params", handleParams);
  server.on("/api", handleApi);

  // Handler per richieste non trovate
  server.onNotFound(handleNotFound);

  // Avvia il server
  server.begin();
  Serial.println("Server HTTP avviato");
}

void loop() {
```

```
  // Gestione continua delle richieste client
  server.handleClient();

  // Piccolo ritardo per stabilità
  delay(2);
}
```

Esempio client-server

```
// ===== ESEMPIO 1: CLIENT HTTP =====
// Esempio tratto da clientHTTP.ino
#include <WiFi.h>

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

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

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

void clientRequest() {
  // Configura parametri server
  const char* host = "google.com";
  const int httpPort = 80;

  // Crea client WiFi
  WiFiClient client;

  // Tenta la connessione al server
  if (client.connect(host, httpPort)) {
    Serial.println("Connessione al server riuscita");

    // Invia richiesta HTTP GET
    client.print("GET / HTTP/1.1\r\n"         // Linea di richiesta con
metodo GET
                 "Host: www.google.com\r\n"   // Header Host richiesto da
HTTP/1.1
                 "Connection: close\r\n\r\n"); // Chiudi la connessione dopo
la risposta

    // Timeout per la risposta
    unsigned long timeout = millis();
```

```
    while (client.available() == 0) {
      if (millis() - timeout > 5000) {
        Serial.println(">>> Timeout client!");
        client.stop();
        return;
      }
    }

    // Leggi e mostra la risposta
    while (client.available()) {
      String line = client.readStringUntil('\r');
      Serial.print(line);
    }
  }
  else {
    Serial.println("Connessione al server fallita");
  }
}

// ===== ESEMPIO 2: SERVER HTTP BASE =====
// Esempio tratto da HelloServer_http_20-03-25.ino
#include <WebServer.h>
#include <ESPmDNS.h>

WebServer simpleServer(80);

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

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

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

  // Configura mDNS per facilitare l'accesso
  if (MDNS.begin("esp32")) {
    Serial.println("Server mDNS avviato");
  }

  // Gestori (handlers) per le varie rotte
  simpleServer.on("/", []() {
    Serial.println("LED ACCESO");
    simpleServer.send(200, "text/html", "<a href='/inline'>spegni</a>");
  });

  simpleServer.on("/inline", []() {
```

```
    Serial.println("LED SPENTO");
    simpleServer.send(200, "text/html", "<a href='/'>accendi</a>");
  });

  // Gestore per pagine non trovate (404)
  simpleServer.onNotFound([]() {
    String message = "File Non Trovato\n\n";
    message += "URI: " + simpleServer.uri() + "\n";
    message += "Metodo: " + (simpleServer.method() == HTTP_GET ? "GET" :
"POST") + "\n";
    message += "Parametri: " + String(simpleServer.args()) + "\n";

    for (uint8_t i = 0; i < simpleServer.args(); i++) {
      message += " " + simpleServer.argName(i) + ": " + simpleServer.arg(i)
+ "\n";
    }

    simpleServer.send(404, "text/plain", message);
  });

  // Avvia il server
  simpleServer.begin();
  Serial.println("Server HTTP avviato");
}

void loopSimpleServer() {
  // Gestione continua delle richieste client
  simpleServer.handleClient();
  delay(2);
}

// ===== ESEMPIO 3: SERVER HTTP CON AUTENTICAZIONE =====
// Esempio tratto da Login_http_02-04-25.ino
#include <WebServer.h>

WebServer authServer(80);

// Pagine web da pagine.h
String loginPage =
"<!DOCTYPE html>"
"<html>"
"   <body>"
"     <form action='/login.html' method='post'>"
"       <label>Username:</label><input type='text' name='username'><br>"
"       <label>Password:</label><input type='password' name='password'><br>
<br>"
"       <input type='submit' value='Submit'>"
"     </form>"
"   </body>"
"</html>";
```

```
String loginPageError =
"<!DOCTYPE html>"
"<html>"
"   <body>"
"     Username o password errati"
"     <form action='/login.html' method='post'>"
"       <label>Username:</label><input type='text' name='username'><br>"
"       <label>Password:</label><input type='password' name='password'><br>
<br>"
"       <input type='submit' value='Submit'>"
"     </form>"
"   </body>"
"</html>";

String adminPage =
"<!DOCTYPE html>"
"<html>"
"   <body>"
"     Pannello Admin"
"   </body>"
"</html>";

String clientPage =
"<!DOCTYPE html>"
"<html>"
"   <body>"
"     Pannello Client"
"   </body>"
"</html>";

// Handler pagina home
void handleHome() {
  authServer.send(200, "text/html", loginPage);
}

// Handler autenticazione
void handleLogin() {
  bool usernameAdmin = false;
  bool passwordAdmin = false;
  bool username = false;
  bool password = false;

  // Analisi dei parametri inviati dal form
  for (uint8_t i = 0; i < authServer.args(); i++) {
    if (authServer.argName(i) == "username" && authServer.arg(i) == "admin")
      usernameAdmin = true;
    if (authServer.argName(i) == "password" && authServer.arg(i) == "1234")
      passwordAdmin = true;
    if (authServer.argName(i) == "username" && authServer.arg(i) ==
```

```
"client")
      username = true;
    if (authServer.argName(i) == "password" && authServer.arg(i) == "pass")
      password = true;
  }

  // Verifica delle credenziali e reindirizzamento
  if (usernameAdmin && passwordAdmin) {
    authServer.send(200, "text/html", adminPage);
  }
  else if (username && password) {
    authServer.send(200, "text/html", clientPage);
  }
  else {
    authServer.send(200, "text/html", loginPageError);
  }
}

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

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

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

  // Configura gli handler delle richieste
  authServer.on("/", handleHome);
  authServer.on("/index.html", handleHome);
  authServer.on("/login.html", handleLogin);

  // Handler per pagine non trovate
  authServer.onNotFound([]() {
    String message = "File Non Trovato\n\n";
    message += "URI: " + authServer.uri() + "\n";
    message += "Metodo: " + (authServer.method() == HTTP_GET ? "GET" :
"POST") + "\n";
    message += "Parametri: " + String(authServer.args()) + "\n";

    for (uint8_t i = 0; i < authServer.args(); i++) {
      message += " " + authServer.argName(i) + ": " + authServer.arg(i) +
"\n";
    }

    authServer.send(404, "text/plain", message);
  });
```

```
  // Avvia il server
  authServer.begin();
  Serial.println("Server HTTP con autenticazione avviato");
}

void loopAuthServer() {
  // Gestione continua delle richieste client
  authServer.handleClient();
  delay(2);
}
```