

Architettura REST con ESP32

Alessandro Privitera

TPS – Prof. Cristiano Tessarolo

Anno Scolastico 2024/2025

Cosa abbiamo visto finora



Ripasso: HTTP con ESP32

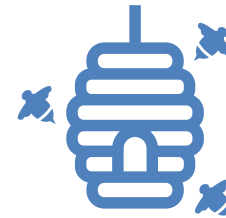
Creazione di webserver con ESP32

- Gestione richieste GET e POST

- Lettura di parametri da URL

- Invio di pagine HTML

- Client HTTP per google.com



Ora: Come organizzare
meglio le nostre API?

Cos'è REST

REST: REpresentational State Transfer

- Architettura per API web
- Usa il protocollo HTTP che già conosciamo
- Organizza le funzionalità come "risorse"

Ogni risorsa ha:

- Un URI univoco
- Operazioni standard (GET, POST, PUT, DELETE) → CRUD
- Rappresentazioni in JSON/XML (formati dati)

Esempio pratico - Dal nostro HelloServer

Dal codice esistente...

- `server.on("/", handleRoot);`
- `server.on("/inline", handleInline);`

...a REST:

- `GET /led/status` -> Leggi stato LED
- `POST /led/on` -> Accendi LED
- `POST /led/off` -> Spegni LED
- `PUT /led/toggle` -> Cambia stato LED

Principi REST per ESP32

- I 4 principi REST applicati a ESP32
 - 1. Risorse come URI
 - /sensors/temperature -> Sensore temperatura
 - /actuators/relay1 -> Relè 1
 - 2. Metodi HTTP standard
 - GET -> Leggi stato
 - POST -> Modifica stato
 - PUT -> Sostituisci configurazione
 - 3. Rappresentazioni JSON
 - {"temperature": 25.4, "unit": "C"}
 - 4. Stateless -> Ogni richiesta è indipendente

Esempio Login REST

- Dal Login tradizionale a REST
 - Tradizionale:
 - `server.on("/index.html", homePage);`
 - `server.on("/login.html", login);`
 - REST:
 - `GET /api/users/session` -> Verifica se loggato
 - `POST /api/auth/login` -> Login con JSON
 - `POST /api/auth/logout` -> Logout
 - `GET /api/users/me` -> Info utente corrente

Codice ESP32 REST – Base (1)

```
#include <ArduinoJson.h>
```

Endpoint REST

- `server.on("/api/sensors/temperature", HTTP_GET, handleGetTemp);`
- `server.on("/api/actuators/led", HTTP_POST, handleLedControl);`
- `server.on("/api/config", HTTP_PUT, handleUpdateConfig);`

Codice ESP32 REST – Base (2)

```
void handleGetTemp()
```

```
StaticJsonDocument<200>  
doc;
```

- doc["temperature"] = readTemperature();
- doc["unit"] = "celsius";
- String response;
- serializeJson(doc, response);
- server.send(200, "application/json", response);
- }

Gestione JSON con ESP32

```
void handleLedControl() {  
  StaticJsonDocument<200> doc;  
  bool state = doc["state"];  
  // switch LED state  
  digitalWrite(LED_PIN, state ? LOW :  
    HIGH);  
  StaticJsonDocument<200> response;  
  response["success"] = true;  
  response["ledState"] = state;  
  String jsonResponse;  
  serializeJson(response, jsonResponse);  
  server.send(200, "application/json",  
    jsonResponse);  
}
```

Struttura URI REST

- Organizzazione gerarchica
 - /api/
 - └─ devices/ # Collezione dispositivi
 - | └─ temperature # Singolo dispositivo
 - | └─ humidity
 - └─ actuators/ # Collezione attuatori
 - | └─ led/1 # LED specifico
 - | └─ relay/2 # Relè specifico
 - └─ config/ # Configurazioni
 - └─ network
 - └─ sensors

Codici HTTP per IoT

- Codici di risposta utili per ESP32
 - 200 OK -> Operazione riuscita
 - 201 Created -> Nuovo dispositivo aggiunto
 - 400 Bad Request -> Comando non valido
 - 401 Unauthorized-> Autenticazione richiesta
 - 404 Not Found -> Sensore non trovato
 - 500 Error -> Errore hardware ESP32
- Esempio:

```
if (sensorAvailable()) {  
    server.send(200,  
    "application/json", data);  
} else {  
    server.send(404,  
    "application/json",  
    "{\"error\":\"Sensor not  
found\"}");  
}
```

Esempio completo - Sistema IoT

```
// Sistema di controllo casa con REST
void setupRESTEndpoints() {
  server.on("/api/status", HTTP_GET,
    getSystemStatus);
  server.on("/api/lights", HTTP_GET,
    getAllLights);
  server.on("/api/lights/room1",
    HTTP_POST, controlLight);
  server.on("/api/sensors/temperature",
    HTTP_GET, getTemperature);
  server.on("/api/sensors/humidity",
    HTTP_GET, getHumidity);
  server.on("/api/config/network",
    HTTP_PUT, updateNetwork);
}
```

Vantaggi REST con ESP32

- Perché usare REST con ESP32?
 - API standard -> Facili da integrare
 - Struttura chiara -> Codice più manutenibile
 - JSON nativo -> Compatibile con app/web
 - Scalabile -> Aggiungi dispositivi facilmente

Questions?

