IIS Di Vittorio Lattanzio (RM)

Dispensa di TPSIT — Metodi HTTP

Classe 4° — A.S. 2025

Docente: **Salvatore Capolupo**

salvatore.capolupo@divittoriolattanzio.edu.it

Spiegazione dei principali metodi HTTP e dimostrazioni pratiche in HTML5 e JavaScript.

Aggiornato al 11 ottobre 2025

Indice

1 Introduzione

In genere l'uso di linguaggi procedurali come Java, C++, PHP ... permette di creare moduli software e/o funzioni in grado di riprodurre operazioni ad alto livello. Ogni funzione va scritta nel minimo dettaglio, sfruttando i parametri di ingresso e producendo una variabile singola o un array di n uscite. Questo approccio può essere integrato con la chiamata di funzioni disponibili esternamente, per fare uso di quella che un tempo veniva chiamata RPC (Remote Procedure Call).

Dalla programmazione procedurale/a oggetti a quella basata su RESTful API

Poniamo di voler creare una calcolatrice con funzioni avanzate di calcolo, e per semplicità, consideriamo un singolo programmatore. È evidente che non ho il tempo di progettare ogni funzione (radici cubiche, medie ponderate con molti numeri, ecc.) e rischierei di dover gestire troppo codice, reinventando la ruota. Grazie all'approccio basato sulle RESTful API posso, in alternativa, richiamare alcune funzioni già pronte, su un "server matematico" esterno, inviare le operazioni che mi servono e ottenere i risultati, ricombinandoli come mi servono. È il caso delle funzioni più complicate come, ad esempio, radice cubica, media ponderata, ecc.

L'uso di RESTful API - RESTful è un gioco di parole tra "*restful*" (rilassante) e l'acronimo "*REpresentational State Transfer*", mentre API sta per *Application Programming Interface* - è un pattern di programmazione che presuppone che l'applicazione abbia accesso alla rete, e per quanto i protocolli utilizzabili siano numerosi il più comodo rimane senza dubbio HTTP(S).

Il protocollo **HTTPS** (HyperText Transfer Protocol Secure) è il modo in cui i client (esempio: il browser Chrome) e appositi server (programmati adeguatamente, di solito in PHP) comunicano tra loro.

Ogni richiesta HTTP fa uso di un **metodo** per indicare che tipo di operazione si vuole effettuare su una risorsa (ad esempio un file, un record o una pagina web).

I metodi delle REST API si riconducono sempre a questi quattro:

- **GET** — Ottiene dati dal server.
- **POST** — Invia dati al server.
- **PUT** — Aggiorna una risorsa già presente.
- **DELETE** — Elimina una risorsa.

In questo contesto per dati intendiamo dati in formato JSON, mentre per risorsa possiamo avere qualsiasi contenuto strutturato di interesse per l'utente (le previsioni del tempo,.

Il formato JSON è un formato di scambio dati che viene rappresentato come "insiemi" di informazioni aggregate, come se fossero le righe di una tabella.

Un primo esempio potrebbe essere il seguente.

```json
{
  "studente": {
    "nome": "Pierino",
    "cognome": "Bianchi",
    "classe": "5OPS",
    "voti": [2, 2.5, 10, 3],
    "is_promosso": false
  }
}
```

Listing 1: Esempio di formato JSON

Se volessimo rappresentare una classe in formato JSON, avremmo invece qualcosa del genere:

Esempio di file JSON per una classe con tre alunni

```json
{
  "classe": "5OPS",
  "materia": "TPSIT",
  "docente": "Prof. Capolupo",
  "alunni": [
    {
      "ID": 1,
      "nome": "Pierino",
      "cognome": "Bianchi",
      "voti": [2, 2.5, 10, 3],
    },
    {
      "ID": 2,
      "nome": "Giulio",
      "cognome": "Rossi",
      "voti": [9, 9, 10],
      "media": 9.3
    },
    {
      "ID": 3,
      "nome": "Maria",
      "cognome": "Blu",
      "voti": [6, 7, 6.5],
    }
  ]
}
```

Attenzione: JSON non fa controlli sulla correttezza dei dati, come è possibile notare osservando che 1) la media non è sempre presente 2) qualora sia presente non è detto che sia corretto. Questa caratteristica è stata voluta dai progettisti per mantenere semplice e leggero il formato, ma poi naturalmente sarà il programmatore a dover verificare la correttezza dei dati.

1.1 1. Metodo GET

Serve per **richiedere** informazioni da una risorsa senza modificarla. È usato, per esempio, per caricare una pagina web o ricevere dati da un'API. Tutti gli esempi che seguono utilizzano il servizio di test pubblico `jsonplaceholder.typicode.com`, utile per esercitarsi senza modificare dati reali.

1.2 2. Metodo POST

Viene utilizzato per **inviare** informazioni al server. È tipico nei moduli HTML: i dati inseriti dall'utente vengono inviati per essere elaborati.

1.3 3. Metodo PUT

Serve per **aggiornare o sostituire** una risorsa già esistente. È usato soprattutto nelle API REST per aggiornare record.

1.4 4. Metodo DELETE

Consente di **rimuovere** una risorsa sul server, come un record o un file.

In sintesi:

Esempi di richieste HTTP in formato JSON

```
GET    /path/studenti/1 #ottieni informazioni sullo studente con ID=1
POST   /path/studenti   # qui si assume che i dati inviati siano
   impliciti  (per motivi di sicurezza: potrebbero essere una password
   !)
PUT    /path/studenti/1 # anche qui si assume che i dati da aggiornare
    siano impliciti
DELETE /path/studenti/2 # cancella lo studente con ID=2
```

La forma effettiva della sintassi naturalmente dipenderà dal server e dalle funzionalità che mette a disposizione, ovvero che ha "esposto" al pubblico. Molte funzioni potrebbero funzionare solo previa autenticazione del client, oppure essere a pagamento.

2 Esempi Pratici in HTML5 e JavaScript

Gli esempi seguenti mostrano come i metodi HTTP possono essere simulati con codice **JavaScript** e un'interfaccia minima in **HTML5**. A scopo di esempio, ci collegheremo con un server che invia JSON di prova con contenuti casuali (lorem ipsum, un segnaposto testuale molto usato dai grafici).

Esempio 1 — Metodo GET

```html
<!DOCTYPE html>
<html lang="it">
<head>
  <meta charset="UTF-8">
  <title>Esempio GET</title>
  <script>
    async function eseguiGET() {
      const risposta = await fetch('https://jsonplaceholder.typicode.
          com/posts/1');
      const dati = await risposta.json();
      document.getElementById("output").innerText =
        "Titolo: " + dati.title + "\nContenuto: " + dati.body;
    }
  </script>
</head>
<body>
  <h2>Richiesta GET</h2>
  <button onclick="eseguiGET()">Esegui una GET</button>
  <pre id="output"></pre>
</body>
</html>
```

Listing 2: Richiesta GET in HTML e JavaScript

Esempio 2 — Metodo POST (invio dati al server)

```html
<!DOCTYPE html>
<html lang="it">
<head>
  <meta charset="UTF-8">
  <title>Esempio POST</title>
  <script>
    async function inviaDati() {
      // Preleva i dati dal form
      const nome = document.getElementById("nome").value;
      const voto = document.getElementById("voto").value;

      // Esegue la chiamata POST
      const risposta = await fetch('https://jsonplaceholder.typicode.
          com/posts', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          studente: nome,
          voto: voto
        })
      });

      // Visualizza il risultato restituito dal server
      const dati = await risposta.json();
      document.getElementById("output").innerText =
        "Dati inviati correttamente:\n" + JSON.stringify(dati, null,
            2);
    }
  </script>
</head>
<body>
  <h2>Invio dati con POST</h2>
  <form onsubmit="event.preventDefault(); inviaDati();">
    Nome: <input type="text" id="nome" required><br>
    Voto: <input type="number" id="voto" min="0" max="10" required><br
        ><br>
    <button type="submit">Invia al server</button>
  </form>
  <pre id="output"></pre>
</body>
</html>
```

Listing 3: Invio dati con POST in HTML5 e JavaScript

Esempio 3 — Metodo PUT (aggiornamento di una risorsa esistente)

```html
<!DOCTYPE html>
<html lang="it">
<head>
  <meta charset="UTF-8">
  <title>Esempio PUT</title>
  <script>
    async function aggiornaStudente() {
      const id = document.getElementById("id").value;
      const nuovoVoto = document.getElementById("voto").value;

      const risposta = await fetch(`https://jsonplaceholder.typicode.com/posts/${id}`, {
        method: 'PUT',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          id: id,
          voto: nuovoVoto
        })
      });

      const dati = await risposta.json();
      document.getElementById("output").innerText =
        "Risorsa aggiornata:\n" + JSON.stringify(dati, null, 2);
    }
  </script>
</head>
<body>
  <h2>Aggiornamento con PUT</h2>
  <form onsubmit="event.preventDefault(); aggiornaStudente();">
    ID studente: <input type="number" id="id" min="1" required><br>
    Nuovo voto: <input type="number" id="voto" min="0" max="10" required><br><br>
    <button type="submit">Aggiorna</button>
  </form>
  <pre id="output"></pre>
</body>
</html>
```

Listing 4: Aggiornamento dati studente con PUT

Esempio 4 — Metodo DELETE (eliminazione)

```html
<!DOCTYPE html>
<html lang="it">
<head>
  <meta charset="UTF-8">
  <title>Esempio DELETE</title>
  <script>
    async function eliminaStudente() {
      const id = document.getElementById("id").value;

      const risposta = await fetch(`https://jsonplaceholder.typicode.com/posts/${id}`, {
        method: 'DELETE'
      });

      if (risposta.ok) {
        document.getElementById("output").innerText =
          `Studente con ID ${id} eliminato (stato ${risposta.status}).`;
      } else {
        document.getElementById("output").innerText =
          `Errore nella richiesta: ${risposta.status}`;
      }
    }
  </script>
</head>
<body>
  <h2>Eliminazione con DELETE</h2>
  <form onsubmit="event.preventDefault(); eliminaStudente();">
    ID studente: <input type="number" id="id" min="1" required><br><br>
    <button type="submit">Elimina</button>
  </form>
  <pre id="output"></pre>
</body>
</html>
```

Listing 5: Eliminazione dati studente con DELETE

3 Esercizi proposti

1. **Esercizio 1 — Elenco utenti semplice**
   Scrivi una pagina HTML che effettui una richiesta `GET` a

`https://jsonplaceholder.typicode.com/users/1`

e mostri i campi name e email dell'utente.

2. **Esercizio 2 — Elenco utenti innestato**
   Recupera in JS tutti gli utenti da `/users` e mostrali in una tabella HTML con nome e address (attenzione: è un campo innestato a due livelli)

3. **Esercizio 3 — Dettaglio utente specifico**

   Aggiungi in HTML un campo di input per inserire un ID, poi invia una `GET` su `/users/{id}` e mostra i dati dell'utente corrispondente.

4. **Esercizio 4 — Creazione con POST**
   Realizza un form HTML con i campi `title` e `body`. Al click su "Aggiungi", invia una richiesta `POST` a `/posts` con i dati in formato JSON, poi mostra la risposta.

5. **Esercizio 5 — Aggiornamento con PUT**
   Permetti di modificare il titolo di un post già esistente (es. ID=1). Invia una richiesta `PUT` a `/posts/1` con un nuovo titolo e visualizza il risultato.

6. **Esercizio 6 — Eliminazione con DELETE**
   Crea un campo dove l'utente può inserire un ID di post. Invia una `DELETE` a `/posts/{id}` e mostra un messaggio di conferma se lo stato della risposta è `200` o `204`.

7. **Esercizio 7 — Elenco e dettagli combinati**
   Mostra la lista di tutti i post (`/posts`) e, cliccando su un titolo, effettua una nuova `GET` a `/posts/{id}` per visualizzare il contenuto completo sotto la lista.

8. **Esercizio 8 — Commenti di un post**
   Crea un campo per inserire un ID di post, poi esegui una `GET` a `/posts/{id}/comments` e mostra in elenco tutti i commenti con nome ed email.

9. **Esercizio 9 — Gestione asincrona avanzata**
   Realizza una funzione che esegua due `fetch` in parallelo: la prima per `/users`, la seconda per `/posts`. Dopo che entrambe sono completate, mostra il numero totale di utenti e di post recuperati.

10. **Esercizio 10 — Interfaccia CRUD simulata**
    Progetta una mini interfaccia con quattro pulsanti: `Leggi`, `Aggiungi`, `Aggiorna`, `Elimina`. Ogni pulsante deve inviare rispettivamente una `GET`, `POST`, `PUT` e `DELETE` alle risorse di `jsonplaceholder.typicode.com/posts`. Mostra in un riquadro i risultati delle operazioni.